REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* 19-02-2004	**2. REPORT TYPE** Final Report	**3. DATES COVERED *(From – To)*** 01-Dec-00 - 19-Feb-04

4. TITLE AND SUBTITLE

Formal Methods for Information Protection Technology
Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks
Part II

5a. CONTRACT NUMBER
ISTC Registration No: 1994p

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Professor I.V. Kotenko Ph.D.

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
St. Petersburg Institute For Informatics & Automation of the Russian Academy of Sciences
39, 14th Liniya
St. Petersburg 199178
Russia

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 BOX 14
FPO 09499-0014

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
ISTC 00-7035

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited. (approval given by local Public Affairs Office)

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The use of open computer networks as an environment for exchange of information across the globe in distributed applications requires improved security measures on the network, in particular, to information resources used in applications. Integrity, confidentiality and availability of the network resources must be assured. To detect and suppress different types of computer unauthorized intrusions, modern network security systems (NSS) must be armed with various protection means and be able to accumulate experience in order to increase its ability to front against known types of intrusions, and to learn new types of intrusions. The project will perform three main tasks.
1. Develop a mathematical model and a tool that simulates various coordinated intrusion scenarios against computer networks;
2. Develop the mathematical foundations, architecture, and principles of implementation of autonomous-software-tool technology implementing the learning system for intrusion detection;
3. Develop the fundamentals, architecture and software for the computer security system based on multi-level encoding for information protection in mass application.
To detect and suppress different types of computer intrusions, modern NSS must be able to accumulate experience in order to increase its ability to front against known type of attacks/intrusions and to learn unknown simple and complex, local and distributed types of attacks. This requires the use of a powerful intelligent learning subsystem (LS) in NSS. That is why the second task of the project concerns to the development of the formal model, architecture, and software prototype of the autonomous intelligent learning system for detection of the attacks/intrusions against computer network.

15. SUBJECT TERMS
EOARD, Mathematical & Computer Sciences, Computer Systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT UNCLAS	**b. ABSTRACT** UNCLAS	**c. THIS PAGE** UNCLAS	UL		/Signed/PAUL LOSIEWICZ, Ph. D.
					19b. TELEPHONE NUMBER *(Include area code)* +44 20 7514 4474

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39-18

ST. PETERSBURG INSTITUTE FOR INFORMATICS AND AUTOMATION

EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT (EOARD)

Project #1994 P

Formal Methods for Information Protection Technology

Final Report

Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks

Part II

Principal Investigator of Task 2
Leading Researcher of the Intelligent Systems Laboratory of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg
November 2003

ST. PETERSBURG INSTITUTE FOR INFORMATICS AND AUTOMATION

EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT (EOARD)

Final Report

Project # 1994P

Task 2: Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks

Part II

Principal Investigator of Task 2
Leading Researcher of the Intelligent Systems Laboratory of SPIIRAS
Ph.D. Professor I.V. Kotenko

St. Petersburg

November 2003

Contents

Chapter 4. Case Study Description

Abstract. The focus of the Chapter is specification of the case study used for validation of the developed software prototypes of the components of Multi-agent IDLS. It describes categories and instances of attacks used in the case study, data sources used and generic data structures representing data of the selected sources. Additionally, the Chapter specifies instances of data structures representing training and testing data sets used in data mining and KDD procedures and demonstrates examples of training and testing instances as they are used in learning procedures.

4.1. Description of Attacks Included in Case Study

To generate training and testing data we selected *four types of attack categories* ([ideval-99], [Das-00], [Kendall-99], [Lippmann *et al*-99], [Lippmann *et al*-00], [Korba-00], [Haines et al-01], [Stolfo *et al*-00], [McClure et al-01], [Scambray et al-01], [Mahoney-03]):

- Probing – surveillance and other probing, i.e. testing a potential target to gather information (e.g., port scanning).
- Remote to local (R2L) – unauthorized access from a remote machine, i.e. attacks in which an unauthorized user is able to bypass normal authentication and execute commands on the target (e.g. via guessing password).
- Denial of service (DOS) – attacks which prevent normal operation, such as causing the target host or server to crash, or blocking network traffic.
- User to root (U2R) – unauthorized access to local superuser (root) privileges.

Besides attack traffic, we generated *background normal traffic*. To form normal traffic we applied network consisting of three hosts including server, client 1 imitating normal work and attacks and client 2 imitating only normal work. The server and client 1 used Windows XP, and client 2 – Windows 2000. We initialized on each host three or four services (FTP, http, netbios, terminal services (3369 tcp)).

The exemplars of attacks selected for each of the attack category are represented in the Tab.4.1. For attack generation we used the components of Attack simulator developed by authors of the Project ([Gorodetski *et al*-02b], [Gorodetski *et al*-02a]) as well as well known utilities (nmap, PipeUpAdmin, etc.).

Table 4.1. Attack categories and exemplars used in case study

#	Attack category	Attack exemplar
1	**Probing**	SYN-scan
2	**R2L**	FTP-crack attack
3	**DoS**	SYN flood
4	**U2R**	PipeUpAdmin

Probing attacks

The essence of the *probing attack* consists in transmission of inquiries to the network services of hosts and analysis of the responses from them. The purpose of the attack is detection of used hosts, protocols, accessible ports of network services, creation rules of the connections identifiers, definition of the active network services, selection of the users' identifiers and passwords.

Examples of probing attacks [Mahoney-03]:

- *Port scans* – testing ports with listening servers. Tools such as NMAP (Fyodor, 2003) and HPING (Sanfilippo, 2003) use sophisticated techniques to make scans hard to detect, for example, scanning with RST or FIN packets (which are less likely to be logged), or using slow scans to defeat an IDS looking for a burst of packets to a range of ports.
- *IP sweep* – testing a range of IP addresses with *ping* to determine which ones are alive. Another way to gather a list of potential targets is to spoof a zone transfer request to a DNS server, as is done by the *ls* command in NSLOOKUP.

- *Fingerprinting* – determining the operating system version of the target based on idiosyncrasies in responses to unusual packets, such as TCP packets with the reserved flags set. This method, implemented by QUESO and NMAP, distinguishes among hundreds of operating system versions using only 7 packets (Fyodor, 1998).
- *Vulnerability testing* – Network administration tools such as SATAN (Farmer & Venema, 1993), SAINT (Kendall, 1998), MSCAN (Kendall, 1998), and NESSUS (Deraison, 2003) test for a wide range of vulnerabilities. These tools serve the dual purpose of allowing network administrators to quickly test their own systems for vulnerabilities, and allowing attackers to test someone else's system. NESSUS is open source, uses a scripting language and has an extensive library of tests, which is updated as new vulnerabilities are discovered. As of February 2003, NESSUS tests for 1181 vulnerabilities.
- *Inside sniffing* – An attacker with physical access to a broadcast medium such as Ethernet, cable TV, or wireless, could sniff traffic addressed to others on the local net. Many protocols such as telnet, FTP, POP3, IMAP, and SNMP transmit passwords unencrypted.

For generation of training and testing data in our case study we used TCP SYN scanning (SYN-scan) attacks. TCP SYN scanning (SYN-scan) can be used for identification of active hosts (it is living host scanning) or active services (ports) (it is port scanning). In the first case an attacker sends SYN packets to the same service port to many different target hosts. Often the target hosts are queried in a systematic, orderly fashion and the attacker sends the probes very frequently. In the second case SYN packets are sent to many different ports to the same target hosts. If from the port the confirmation packet SYN/ACK is received, the port listens; if the packet RST/ACK is received, the port does not listen; if the answer is absent, the host does not work.

Two kinds of SYN-scan used for generation of training and testing data: TCP connect scan and TCP Half SYN scan. TCP connect scan are based on full connection realization. Half scan is used for a stealth detection of the open ports on a host. The packets initializing a connection are dispatched. When responses come back, a connection is dropped, i.e. a packet having a flag RST is transferred.

R2L attacks

R2L attacks always exploit application protocols to gain control over the target. Kendall [Kendall-99] describes several attacks, which can be grouped as follows:

- *Password guessing* – Many users tend to choose weak or easily guessed passwords. An attack could try common passwords such as *guest*, the user name, or no password. If this fails, an attacker could use a script to exhaustively test every word in a dictionary. Any service requiring a password is vulnerable, for example, telnet, FTP, POP3, IMAP, or SSH.
- *Server vulnerability* – An attacker exploits a software bug to execute commands on the target, often as root. For example, buffer overflow vulnerabilities have been discovered in *sendmail* (SMTP), *named* (DNS), and *imap*. Other bugs may allow a command to be unwittingly executed. For example, the PHF attack exploits a badly written CGI script installed on default on an old version of *apache*. The following HTTP command will retrieve the password file on a vulnerable server: `GET /cgi-bin/phf?Qalias=x%0a/usr/bin/ypcat%20passwd`
- *Configuration error* – An attacker exploits an unintended security hole, such as exporting an NFS partition with world write privileges. One common error is setting up an open X server (using the command `xhost +`) when running a remote X application. The *xlock* attack scans for open X servers, then displays a fake screensaver which prompts the user to enter a password, which is then captured. *xsnoop* does not display anything; it merely captures keystrokes.
- *Backdoors* – Once a host has been compromised, the attacker will usually modify the target to make it easier to break in again. One method is to run a server such as *netcat*, which can listen for commands on any port and execute them (Armstrong, 2001).

For generation of training and testing data in our case study we have used Password Guessing (Cracking) attacks, including FTP-crack attacks. This attack class has a goal to gain an access (either as a user or as a root) to the target host. An attacker (who does not have an account on the target host) sends packets to that host over the network, guessing a password for a valid user. Frequently, this attack is realized by a simple dictionary (i.e. by using of simple variants of the account name). It can be done over many services (telnet, FTP, pop, etc.).

DoS attacks

Denial of service attacks can target a server, a host, or a network. These either flood the target with data to exhaust resources, or use malformed data to exploit a bug [Mahoney-03]. The attack "Denial of Service" is designed to prevent legitimate users from using a system. Traditional Denial of Service attacks are done by exploiting a buffer overflow, exhausting system resources, or exploiting a system bug that results in a system that is no longer functional.

Kendall [Kendall-99] gives the following examples, all of which are used in the IDEVAL test set [ideval-99]:

- *Apache2* – Some versions of the *apache* web server will run out of memory and crash when sent a very long HTTP request. Kendall describes one version in which the line "User-Agent: sioux" is repeated 10,000 times.
- *Back* – Some versions of *apache* consume excessive CPU and slow down when the requested URL contains many slashes, i.e. "GET ////////////...".
- *Land* – SunOS 4.1 crashes when it receives a spoofed TCP SYN packet with the source address equal to the destination address.
- *Mailbomb* – A user is flooded with mail messages.
- *SYN flood (Neptune)* – A server is flooded with TCP SYN packets with forged source addresses. Because each pending connection requires saving some state information, the target TCP/IP stack can exhaust memory and refuse legitimate connections until the attack stops.
- *Ping of death* – Many operating systems could be crashed (in 1996 when the exploit was discovered) by sending a fragmented IP packet that reassembles to 65,536 bytes, one byte larger than the maximum legal size. It is called "ping of death" because it could be launched from Windows 95 or NT with the command "ping –l 65510 *target*".
- *Process table* – An attacker opens a large number of connections to a service such as *finger*, POP3 or IMAP until the number of processes exceeds the limit. At this point no new processes can be created until the target is rebooted.
- *Smurf* – An attacker floods the target network by sending ICMP ECHO REQUEST (*ping*) packets to a broadcast address (x.x.x.255) with the spoofed source address of the target. The target is then flooded with ECHO REPLY packets from multiple sources.
- *Syslogd* – The *syslog* server, which could be used to log alarms remotely from an IDS, is crashed by sending a spoofed message with an invalid source IP address. Due to a bug, the server crashes when a reverse DNS lookup on the IP address fails.
- *Teardrop* – Some operating systems (Windows 95, NT, and Linux up to 2.0.32) will crash when sent overlapping IP fragments in which the second packet is wholly contained inside the first. This exploits a bug in the TCP/IP stack implementation in which the C function *memcpy*() is passed a negative length argument. The argument is interpreted as a very large unsigned number, causing all of memory to be overwritten.
- *UDP storm* – This attack sets up a network flood between two targets by sending a spoofed UDP packet to the *echo* server of one target with the spoofed source address and port number of the *chargen* server of the other target.

For generation of training and testing data in our case study we have used SYN flood attack. The *SYN flood attack* consists of a storm of inquiries on installation of TCP-connections. In a TCP three–steps handshake, the server responds to a client's initial SYN packet by sending a SYN-ACK. The server is waiting for another ACK from the client up to the connection becomes "established". Generally, this is not a problem if the server is only waiting for single or a few connections to complete the handshake. However, the server's queue for holding "waiting for connections to be completed" is of finite size. Thus, if an attacker send many spoofed SYN packets from non-existing IP addresses to a single target port on the server then the server's queue would fill up and the server would become temporarily unable to respond to any new service requests. Further, the server will remain in this state until the "waiting to be established" connections are timed out, which can take a minute or two. During this time, legitimate clients will be unable to establish connections with the server on the target port.

U2R attacks

In User to root (U2R) attacks a user with login access is able to bypass normal authentication to gain the privileges of another user, usually root (e.g. various ``buffer overflow" attacks). U2R attacks exploit bugs or misconfigurations in the operating system, for example a buffer overflow or incorrectly set file permissions in a *suid root* program. U2R attacks allow increasing privileges of users to the superuser (for example, SYSTEM).

In our experiments we used the U2R attack "*PipeUpAdmin*". It is based on well known vulnerability of Windows 2000 permitting to increase privileges via prediction of named pipes of Service Control Manager (SCM) [Scambray et al-01]. This vulnerability allows to interactively connected users to get SYSTEM privilege.

The SCM service uses in intraprocess communication unique names of pipes for each executed service. The format of name for such a pipe is as follows: \\.pipe\net\NtControlPipe12 , where 12 is the pipe number. After reading the value of the Register key HKLM\SYSTEM\CurrentControlSet\Control\ServiceCurrent, an attacker can conclude that the next name of the pipe will be \\.\pipe\net\NtControlPipe13 .

In this attack a predictability of a pipe number is used, which is created before creating of a pipe with the same name by service SCM. When the new service is initialized, it will connect to the malefactor's pipe. After this event the malefactor's pipe can use context of security service and he/she get a possibility to execute commands with the SYSTEM privilege if this privilege is used by the service. The utility PipeUpAdmin can be used to use this vulnerability. PipeUpAdmin adds account of the current user to the group of the local administrators. Let us consider an example of this utility usage.

Let the user Vladimir initialize the utility:

```
C:\>pipeupadmin

                          PipeUpAdmin
                 Maceo <maceo@dogmile.com>
                 (C) Copyright 2000-2001 dogmile.com

The ClipBook service is not started. More help is available by typing NET HELPMSG 3521.
Impersonating: SYSTEM
The account: FS-EVIL\vladimir
has been added to the Administrators group.
```

Then the user Vladimir executes command net *localgroup* and finds his name in the group of the local administrators:

```
C:\>net localgroup administrators

Alias name     administrators
Comment        Administrators have complete and unrestricted
               access to the computer/domain

Members
-------------------------------------------------------------------------------
Administrator
vladimir

The command completed successfully.
```

The next action of the user vladimir to get SYSTEM privileges is to log off and again log on. This operation is needed since Windows 2000 has to change security token of the current user to add to it the identifier SID of a new group. The security token can be changed by API function call or exit from the system (log off) and next log on.

4.2. Data Sources and Structures Representing Training and Testing Data

The data sources and generic data structures representing training and testing data described below were partially selected on the basis of compilation of the results published in ([ideval-99], [Lee-99], [Lee *et al*-00a], [Stolfo *et al*-00], [Dokas *et al*-02], [Lazarevic *et al*-03a], etc.) and partially were proposed by authors of the Project.

The motivations of chosen selection are as follows:

1. The objective of the Project is to develop technology for learning of intrusion detection on the basis of multiple heterogeneous data sources, therefore the number of data sources must be no less than two.
2. Different attacks can produce evidences in different sources (levels) and in different sets of features produced on the basis of raw data of each level. This is the reason of using several data structures produced on the basis of the respective raw data in each data source.
3. It is desirable to involve in intrusion detection learning procedure heterogeneous data structures to validate the feasibility of the developed distributed learning technology as well as multi-agent architecture and data fusion model of IDLS. This is the reason of using data structures of both time-based and relational natures.
4. It is desirable to use the same generic data structures for each particular data source. This can made it easier the code writing efforts due to minimization of particular data mining and KDD techniques covering the learning needs.
5. Practically, each attack is developing on a background reflecting normal user activity, this is why it is necessary to use training and testing data representing malicious user activity mixed with normal user activity background.

Actually three data sources are selected, that are network-based (traffic level), host-based (operating system level) and application-based (in particular, FTP-server level). It is supposed that each data source is represented by four *generic data structures*, which are the same for each data source. These data structures correspond to data produced on the basis of processing of raw data. These data structure are as follows:

1. *Time ordered sequence (time series) of values of binary vectors of parameters specifying significant events of raw data of a level (traffic level, OS logs and FTP-server logs).* Graphical explanation of this data structures is given in Fig.4.1. Specialization of this data for each data source is given in the next section.

   In fact, this data structure specifies time series and it is supposed that technique developed specifically for discrete vector time series based on correlation and regression analysis is used.

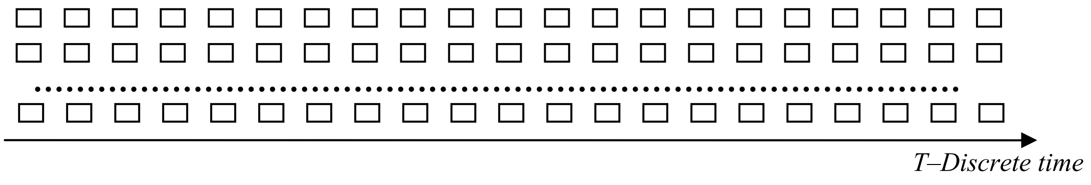
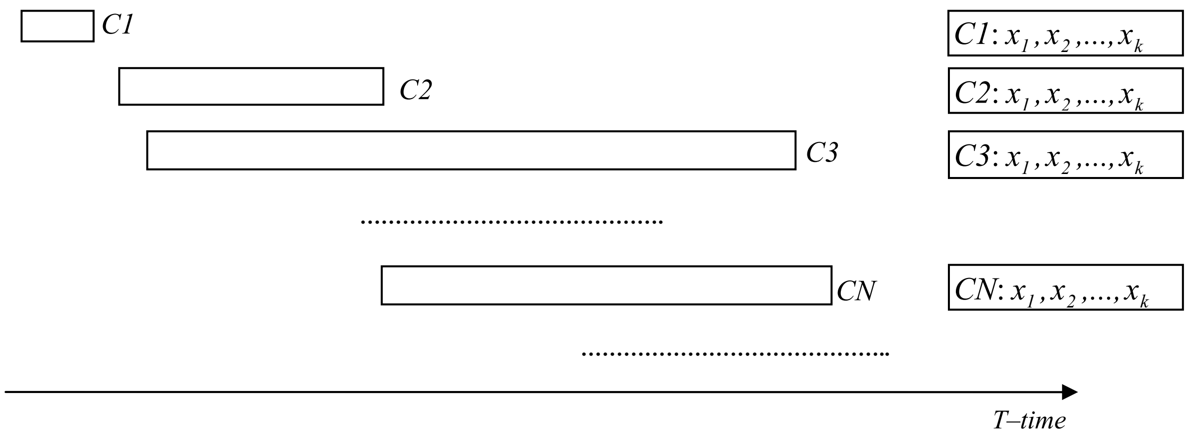


Fig.4.1. Graphical representation of data structures specifying time-ordered binary vectors of significant events of raw data in each data source

2. *Statistical attributes of particular connections (performance of a user) showed in a data source (traffic level, OS logs and FTP-server logs).* Each such slot of data consists of the same attributes for each data source mapped to the time interval of connection appearance. Graphical explanation of this generic data structure is given in Fig.4.2.
3. *Statistical attributes of traffic (users' activity) during the short term time intervals* (2 or 5 second duration as recommended in ([ideval-99], [Lee-99], [Lee *et al*-00a], [Stolfo *et al*-00], [Dokas *et al*-02], [Lazarevic *et al*-03a], etc.). Graphical explanation of this data structure is given in Fig.4.3.



C1
C2
C3
CN
$C1: x_1, x_2, \ldots, x_k$
$C2: x_1, x_2, \ldots, x_k$
$C3: x_1, x_2, \ldots, x_k$
$CN: x_1, x_2, \ldots, x_k$
T–time

Fig.4.2. Graphical representation of data structures specifying statistical attributes of particular connections in each data source mapped to the respective time interval of connection performance

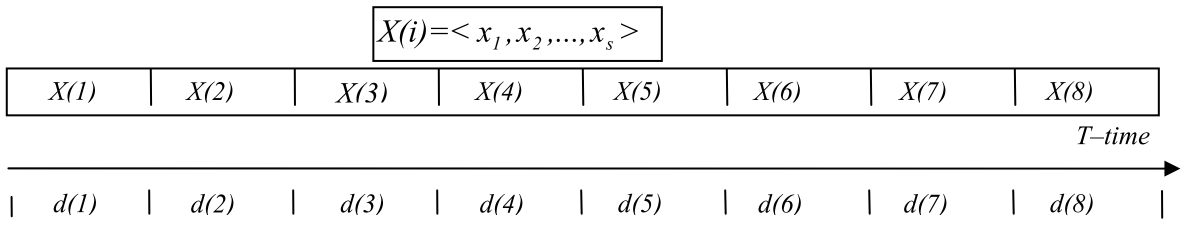


Fig.4.3. Graphical representation of data structures specifying integrated (disregarding particular connections) statistical attributes of a data source (traffic level, OS logs and FTP-server logs) within short term time interval *d(i)* (as a rule*, d=5 sec*)*, i=1,2,…*

4. *Statistical attributes of traffic (users' activity) during the long term time intervals* corresponding to a given number of connections (as a rule*, d=100 connections* that can correspond up to *decades of minutes)* ([ideval-99], [Lee-99], [Lee *et al*-00a], [Stolfo *et al*-00], [Dokas *et al*-02], [Lazarevic *et al*-03a], etc.) or *10-60 min* interval. Graphical explanation of this data structure is given in Fig.4.4.

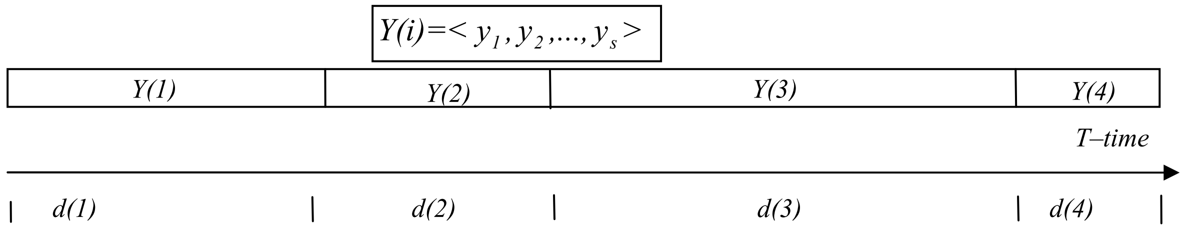


Fig.4.4. Graphical representation of data structures specifying integrated (disregarding particular connections) statistical attributes of a data source (traffic level, OS logs and FTP-server logs) within long term time interval *d(i)* corresponding to a given number of connections (as a rule*, d=100 connections*)*, i=1,2,…*

4.3. Specification of Instances of Data Structures of Different Sources

Let us consider main instances of training and testing data structures of *three sources* chosen: network-based source (traffic level); host-based source (operating system level); application-based source (FTP-server level). Each data source is presented by four generic data structures, which are the same for each data source.

1. Specification of instances of data structures of network-based source (traffic level)

The data structures of network-based source (traffic level) were produced on the basis of processing of tcpdump/windump data. We used for this TCPtrace utility [Tcptrace] and several programs developed by the authors of the Report. Let us consider instances of the data structures of network-based source (traffic level).

Time ordered sequence of values of binary vectors of parameters specifying significant events of network traffic level

We chose the combination of values of the tag *Flag* of TCP packet header as parameters specifying significant events. In this data structure, each network packet is described by binary vector consisted of six values of the following separate flags: U (URG) – urgent pointer is valid; A (ACK) – acknowledgement number is valid; P (PSH) – tells receiver not to buffer the data before passing it to the application; R (RST) – reset (abort) the connection; S (SYN) – synchronize the sequence numbers to establish a connection; F (FIN) – finish of data transmission. Thus, each vector corresponding to certain network event is described by the seven features: Time, U, A, P, R, S, F, where values of the last six attributes are binary (1 or 0). Examples of binary vectors of parameter *Flag* are as follows:

U	A	P	R	S	F	Corresponding integral value of the parameter *Flag*:
0	0	0	0	1	0	S
0	1	0	0	0	0	A
0	1	0	0	1	0	SA

Statistical attributes of particular connections

The list of the parameters (features) describing this data structure is as follows:

Feature	**Description**	**Value Type**
time	Time of connection initialization	Continuous
duration	Length (number of seconds) of the connection	Continuous
connection_status	Status of the connection (0 – Completed; 1 - Not completed; 2 – Reset)	Discrete
num_packets	The common number of packets during connection	Continuous

The key parameters "*time*" and "*duration*" allow for setting a time ordered sequence of attributes characterizing separate connections.

Statistical attributes of network traffic during the short term time intervals (5 second)

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
count_src	Number of connections made by the same source as the current record	Continuous
count_dest	Number of connections made to the same destination as the current record	Continuous
count_serv_src	Number of different services from the same source as the current record	Continuous
count_serv_dest	Number of different services to the same destination as the current record	Continuous

Statistical attributes of network traffic during the long term time intervals (100 connections)

The list of the parameters describing this data structure is as follows:

Feature Name	**Feature description**	**Value Type**
count_src	Number of connections made by the same source as the current record	Continuous
count_dest	Number of connections made to the same destination as the current record	Continuous
count_serv_src1	Number of connections with the same service made by the same source as the current record	Continuous
count_serv_dst1	Number of connections with the same service made to the same destination as the current record	Continuous

2. Specification of instances of data structures of host-based source (operating system level)

The data structures of host-based source (operating system level) were produced on the basis of processing of raw data of operating system log *Security* (for Windows 2000/XP). We used for this processing several programs developed by the authors of the Report. Let us consider instances of the data structures of host-based source (operating system level).

Time ordered sequence of values of binary vectors of parameters specifying significant events on operating system level

The values of binary vectors are signs of appearance (1 or 0) of significant events from operating system log *Security* (for Windows 2000/XP). In this data structure, each OS event is described by binary vector consisting of values of the parameters denoting events numbers significant for attacks generated.

For our case study we chose 10 events numbers significant for the attacks generated: 512 – Window is starting up; 513 – Windows is shutting down; 517 – The audit log was cleared; 528 – Successful Logon; 529 – Logon Failure; Unknown user name or bad password; 530 – Logon Failure; Account logon time restriction violation; 538 – The logoff process was completed for a user; 592 – The virtual address space and the control information necessary for the execution of a program was created; 636 – Security Enabled Local Group Member Added; 680 – Logon attempt.

Thus, each vector corresponding to certain OS event is described by eleven features: Time, 512, 513, 517, 528, 529, 530, 538, 592, 636, 680, where values of last 10 attributes are binary (1 or 0).

Examples of binary vectors describing OS events (without time attribute) are as follows (each string corresponds to particular vector, XXX – any event that is not significant for reflecting attacks generated):

512	513	517	528	529	530	538	592	636	680	Corresponding OS Event
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529

Statistical attributes of particular sessions of a user performance on OS level

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
time	Time of the process initialization	Continuous
duration	Length (number of seconds) of the process execution	Continuous
logged_in	1 - successfully logged in; 0 - otherwise	Discrete
hot	Number of “hot indicators” (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
mem_add	1 - Security enabled local group member added; 0 - otherwise	Discrete

Statistical attributes of OS functioning during the short term time intervals (5 second)

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
hot	Number of “hot indicators” (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
failed_logins	Number of failed login attempts	Continuous
shutt_down	1 - OS shutting down; 0 - otherwise	Discrete

starting_up	1 - OS starting up; 0 - otherwise (we assume that OS event "starting up" marks the beginning of time interval 5 sec)	Discrete
mem_add	1 - Security enabled local group member added; 0 - otherwise	Discrete

Statistical attributes of OS functioning during the long term time intervals (10 – 60 minutes)

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
hot	Number of "hot indicators" (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
Failed_logins	Number of failed login attempts	Continuous
shutt_down	Number of OS shutting down	Continuous
starting_up	Number of OS starting up	Continuous
mem_add	Number of events when security enabled local group member added; 0 - otherwise	Continuous

3. Specification of instances of data structures of application-based source (FTP-server level)

The data structures of application-based source (FTP-server level) were produced on the basis of processing of raw data of FTP-server log. We used for this processing several programs developed by the authors of the Report.

Let us consider instances of the data structures of application-based source (FTP-server level).

Time ordered sequence of values of binary vectors of parameters specifying significant events on FTP-server level

The values of binary vectors are signs of appearance (1 or 0) of significant events from FTP-server log. In this data structure (as for OS level), each event is described by binary vector consisted of several values of the parameters denoting events numbers from FTP-server log (FTP return codes) that are significant for attacks generated.

For our case study we chose preliminarily only 2 events numbers (FTP return codes) significant for the attacks generated: 331 – User name okay, need password; 530 – Not logged in. Thus, each vector corresponding to some FTP event is described by three features: Time, 331, 530, where values of last two attributes are binary (1 or 0). Examples of binary vectors describing FTP events (without time attribute) are as follows (each string corresponds to particular vector, XXX – any event that is not significant for reflecting attacks generated):

331	530	Corresponding FTP Event
0	0	XXX
1	0	331
0	1	530

Statistical attributes of particular sessions of a user performance on FTP-server level

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
time	Time of the process initialization	Continuous
duration	Length (number of seconds) of the process execution	Continuous

hot	Number of "hot indicators" (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
failed_logins	Number of failed login attempts	Continuous
logged_in	1 - successfully logged in; 0 - otherwise	Discrete

Statistical attributes of FTP-server functioning during the short term time intervals (5 second)

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
Hot	Number of "hot indicators" (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
failed_logins	Number of failed login attempts	Continuous
successful_logins	Number of successful logins	Continuous

Statistical attributes of FTP-server functioning during the long term time intervals (10 – 60 minutes)

The list of the parameters describing this data structure is as follows:

Feature	**Description**	**Value Type**
hot	Number of "hot indicators" (e.g., access to system directories, creation and execution of programs, etc.)	Continuous
failed_logins	Number of failed login attempts	Continuous
successful_logins	Number of successful logins	Continuous

4.4. Examples of Training and Testing Data

Let us consider examples of training and testing data for each of four generic data structures of epy three sources. These examples are presented for all five exemplars of attacks concerning to four attacks categories selected. We describe also examples of data fixed for normal users' work. For each data structure we determine only examples of attacks data differing from normal work.

The heading for each exemplar of data is represented in the following format:

Attack category: Attack exemplar.

4.4.1. Examples of Training and Testing Data of Network-based Source (Traffic Level)

1. ***Examples of time ordered sequences of values of binary vectors of parameters specifying significant events of network traffic level***

(1) Probing: SYN-scan

Example 1 (representing only abnormal traffic):

Representation of instances of training and testing data as ordered sequence of integral values of the parameters *Flag*:

S R R S SA R R S R R S R R S R R S SA R R … ,

where sequence S R R describes the state "*port is closed*", and S SA R R – "*port is open*".

This data can be represented as time ordered sequence of binary vectors of parameter *Flag* values:

Number of binary vector	Time	Values of parameter *Flag*						Corresponding integral value of the parameters *Flag*
		U	A	P	R	S	F	
1	16:35:18.555138	0	0	0	0	1	0	S
2	16:35:18.555195	0	0	0	1	0	0	R
3	16:35:19.565206	0	0	0	1	0	0	R

4	16:35:19.591847	0	0	0	0	1	0	S
5	16:35:19.591940	0	1	0	0	1	0	SA
6	16:35:19.592889	0	0	0	1	0	0	R
7	16:35:20.618765	0	0	0	1	0	0	R
…	….	…	…	…	…	…	…	…

To abridge the representation of examples of data, below in this subsection we use only representation of instances of training and testing data as ordered sequences of integral values of the parameters *Flag*.

Example 2 (representing combination of abnormal and normal traffic):
S X R R X S SA R X R S R R S X R R S R R X S SA X R R …
S R X R X X S X SA R X R X S R X R S R X R S R X R S X SA R X R … ,
where X – integral value of the parameters *Flag* reflecting normal traffic.

(2) R2L: FTP-crack

Example 1 (representation of the completed connection reflecting two attempts of password guessing):
S SA A PA PA PA PA PA PA PA PA PA PA PA PA PA PA FA A

Example 2 (representation of the completed connection reflecting three attempts of password guessing):
S SA A PA PA PA PA PA PA PA PA PA PA PA PA PA PA PA PA FA A

(3) DoS: SYN-flood

Example 1 (representation of the attack process when "victim" does not answer):
S S S S S S S S S S S S S S …

Example 2 (representation of the attack process when "victim" answers):
S SA S SA S SA S SA S SA S SA S SA S SA S SA S SA S SA S SA S …

(4) Normal traffic

Example 1:
S SA S PA PA PA SA F PA PA FA PA PA F FA …

2. ***Examples of statistical attributes of particular connections***

For description of examples, we use the following features:

- *Time* - Time of the connection initialization;
- *Duration* - Length (number of seconds) of the connection;
- *connection_status* - Status of the connection (0 – Completed; 1 - Not completed; 2 – Reset);
- *num_packets* - The common number of packets.

(1) Probing: SYN-scan

Example 1:

time	*duration*	*connection_status*	*num_packets*
Wed May 21 16:35:18.555138 2003	0:00:01.010068	2	3
Wed May 21 16:35:19.591847 2003	0:00:01.026917	2	4
Wed May 21 16:35:20.640751 2003	0:00:01.007608	2	3
Wed May 21 16:35:19.791747 2003	0:00:01.026917	2	4

(2) R2L: FTP-crack

Example 1 (number of packets transmitted during different connections is the same):

time	*duration*	*connection_status*	*num_packets*
Tue May 6 17:53:15.359650 2003	0:00:00.181004	0	20
Tue May 6 17:53:15.534327 2003	0:00:00.115962	0	20
Tue May 6 17:53:15.651894 2003	0:00:00.131135	0	20

Example 2 (number of packets transmitted during different connections can be changed, i.e. for one connection one, two or three passwords can be transmitted):

time	*duration*	*connection_status*	*num_packets*
Tue May 6 17:53:15.359650 2003	0:00:00.181004	0	20
Tue May 6 17:53:15.534327 2003	0:00:00.115962	0	15
Tue May 6 17:53:15.651894 2003	0:00:00.131135	0	20
Tue May 6 17:53:15.732317 2003	0:00:00.115962	0	10

In this example in the first connection three password are transmitted, in the second - two, in the third - three and in the forth - one.

(3) DoS: SYN-flood

Example 1:

time	*duration*	*connection_status*	*num_packets*
Fri May 23 17:05:20.232783 2003	0:00:00.000000	1	1
Fri May 23 17:05:20.234642 2003	0:00:00.000000	1	1
Fri May 23 17:05:20.248258 2003	0:00:00.000000	1	1
Fri May 23 17:05:20.252783 2003	0:00:00.000000	1	2
Fri May 23 17:05:20.258642 2003	0:00:00.000000	1	2
Fri May 23 17:05:20.261258 2003	0:00:00.000000	1	2

(4) Normal traffic

Example 1:

time	*duration*	*connection_status*	*num_packets*
Fri May 23 17:05:20.232783 2003	0:00:10.000000	0	10
Fri May 23 17:05:20.234642 2003	0:01:01.000000	0	1000
Fri May 23 17:05:20.248258 2003	0:00:45.000000	0	55

3. ***Examples of statistical attributes of network traffic during the short term time intervals (5 second)***

For description of examples, we use the following features:

- *count_src* - Number of connections made by the same source as the current record;
- *count_dest* - Number of connections made to the same destination as the current record;
- *count_serv_src* - Number of different services from the same source as the current record;
- *count_serv_dest* - Number of different services to the same destination as the current record.

(1) Probing: SYN-scan

Example 1:

count_src	*count_dest*	*count_serv_src*	*count_serv_dest*
2	2	1	1
2	2	1	1
3	3	1	1
2	2	1	1
3	3	1	1

(2) R2L: FTP-crack

Example 1:

count_src	*count_dest*	*count_serv_src*	*count_serv_dest*
5	5	1	1
5	5	1	1
5	5	1	1
6	6	1	1
6	6	1	1

(3) DoS: SYN-flood

Example 1:

count_src	*count_dest*	*count_serv_src*	*count_serv_dest*
750	750	1	1
742	742	1	1
746	746	1	1
736	736	1	1
740	740	1	1
748	748	1	1

(4) Normal traffic

Example 1:

count_src	*count_dest*	*count_serv_src*	*count_serv_dest*
1	2	1	1
3	3	1	1
10	12	3	2
5	6	2	2
3	4	1	1
12	12	3	2

4. *Examples of statistical attributes of network traffic during the long term time intervals corresponding to 100 connections*

For description of examples, we use the following features:

- count_src - Number of connections made by the same source as the current record;
- count_dest - Number of connections made to the same destination as the current record;
- count_serv_src1 - Number of connections with the same service made by the same source as the current record;
- count_serv_dst1 - Number of connections with the same service made to the same destination as the current record.

(1) Probing: SYN-scan

Example 1 (SYN-scan attacks were executed only in several connections, other connections reflect normal traffic):

count_src	Count_des	count_serv_src1	count_serv_dst1
39	46	6	12
50	49	6	12
65	46	9	14
39	42	6	9
63	57	7	13

(2) R2L: FTP-crack

Example 1 (FTP-crack attacks were executed in most of connections):

count_src	Count_dest	count_serv_src1	count_serv_dst1
64	61	26	12
71	79	23	12
79	76	39	14
64	59	26	9
74	83	57	13

(3) DoS: SYN-flood

Example 1:

count_src	count_dest	count_serv_src1	count_serv_dst1
99	99	99	99
98	98	98	98
97	97	97	97
97	97	97	97
95	95	95	95
94	94	94	94

(4) Normal traffic

Example 1:

count_src	count_dest	count_serv_src1	count_serv_dst1
43	83	21	45
50	98	25	52
45	90	16	35
43	87	19	39
42	83	23	46

4.4.2. Examples of Training and Testing Data of Host-based Source (Operating System Level)

1. ***Examples of time ordered sequences of values of binary vectors of parameters specifying significant events on operating system level***

(1) R2L: FTP-crack

Example 1:

Representation of instances of training and testing data as ordered sequence of OS event numbers:

XXX 680 529 XXX XXX 680 529 680 529 XXX 680 529 XXX XXX XXX XXX 680 529,

where XXX – any event that is not significant for reflecting attacks generated;
529 – Logon Failure. Unknown user name or bad password;
680 – Logon attempt.

The corresponding vectors describing OS events (without time attribute) are as follows (each string corresponds to particular vector):

512	*513*	*517*	*528*	*529*	*530*	*538*	*592*	*636*	*680*	*Corresponding OS Event*
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	0	XXX
0	0	0	0	0	0	0	0	0	1	680
0	0	0	0	1	0	0	0	0	0	529

To abridge the representation of examples of data, below in this subsection we use only representation of instances of training and testing data as ordered sequences of OS event numbers.

(2) DoS: SYN-flood

This attack does not result on OS shut down for Windows 2000/XP. So the OS event sequence is the same as for normal traffic.

Example 1:

XXX XXX XXX XXX XXX XXX 530 XXX XXX XXX XXX XXX XXX,

where XXX – any event that is not significant for reflecting attacks generated; 530 – Logon Failure. Account logon time restriction violation.

(3) U2R: PipeUpAdmin

Example 1:

680 528 XXX XXX XXX XXX 592 592 592 592 592 592 592 XXX 636 XXX XXX 538 ,

where XXX – any event that is not significant for reflecting attacks generated; 528 – Successful Logon; 592 – The virtual address space and the control information necessary for the execution of a program was created; 538 – The logoff process was completed for a user; 636 – Security Enabled Local Group Member Added; 680 – Logon attempt.

The first creation of the process under initialization of PipeUpAdmin.exe (corresponding to event 592) produces two sequences of events. Each of these sequences consists of generating three processes («\WINNT\system32\cmd.exe», «\WINNT\system32\net.exe» and «\WINNT\system32\net1.exe»). Thus, six additional events 592 are fixed.

After malefactor cleans OS SECURITY log it will fix the following sequence of events:

517 XXX XXX XXX 538 XXX XXX … ,

where XXX – any event that is not significant for reflecting attacks generated; 517 – The audit log was cleared; 538 – The logoff process was completed for a user.

(4) Normal traffic

Example 1:

XXX XXX XXX XXX XXX XXX 530 XXX XXX XXX XXX XXX XXX

2. ***Examples of statistical attributes of particular sessions of a user performance on OS level***

(1) R2L: FTP-crack

Example 1:

time	*duration*	*logged_in*	*hot*	*mem_add*
17:53:45	0	0	0	0
17:53:45	0	0	0	0
17:53:46	0	0	0	0
17:53:47	0	0	0	0

(2) U2R: PipeUpAdmin

Example 1:

time	*duration*	*logged_in*	*hot*	*mem_add*
7:40:41	0:1:29	1	1	1

(3) Normal traffic

Example 1:

time	*duration*	*logged_in*	*hot*	*mem_add*
17:53:45	5.005	1	6	0
17:53:47	1.654	1	0	0
17:53:54	452.234	1	20	0

3. *Examples of statistical attributes of OS functioning during the short term time intervals (5 second)*

(1) R2L: FTP-crack

Example 1:

hot	*failed_logins*	*shutt_down*	*starting_up*	*mem_add*
0	12	0	0	0
0	14	0	0	0
0	6	0	0	0

(2) U2R: PipeUpAdmin

Example 1:

hot	failed_logins	shutt_down	starting_up	mem_add
2	0	0	0	1
2	0	0	0	0

(3) Normal traffic

Example 1:

hot	failed_logins	shutt_down	starting_up	mem_add
2	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	1	0	0
6	0	0	1	1
0	0	0	0	0
0	1	0	0	0
2	0	0	0	0
0	0	0	0	0

4. *Examples of statistical attributes of OS functioning during the long term time intervals corresponding to 10 – 60 minutes*

(1) R2L: FTP-crack

Example 1:

hot	*failed_logins*	*shutt_down*	*starting_up*	*mem_add*
2	120	0	0	0
12	344	1	1	0
0	248	0	0	0

(2) U2R: PipeUpAdmin

Example 1:

hot	*failed_logins*	*shutt_down*	*starting_up*	*mem_add*
12	1	0	0	1
10	0	0	0	0

(3) Normal traffic

Example 1:

hot	*failed_logins*	*shutt_down*	*starting_up*	*mem_add*
12	0	0	0	0
44	0	0	0	0
20	1	0	0	0
28	0	1	1	0
0	0	0	0	0

4.4.3. Examples of Training and Testing Data of Application-based Source (FTP-server Level)

1. ***Examples of time ordered sequences of values of binary vectors of parameters specifying significant events on FTP-server level***

(1) R2L: FTP-crack

Example 1:

Representation of instances of training and testing data as ordered sequence of FTP event numbers (corresponding to the FTP return codes):

XXX XXX 331 530 XXX XXX XXX 331 530 331 530 331 530 XXX XXX ,

where XXX – any event that is not significant for reflecting attacks generated; 331 – User name okay, need password; 530 – Not logged in.

The corresponding vectors describing FTP events (without time attribute) are as follows (each string corresponds to particular vector):

331	*530*	*Corresponding FTP Event*
0	0	XXX
0	0	XXX
1	0	331
0	1	530
0	0	XXX
0	0	XXX
0	0	XXX
1	0	331
0	1	530
1	0	331
0	1	530
1	0	331
0	1	530
0	0	XXX
0	0	XXX

To abridge the representation of examples of data, below in this subsection we use only representation of instances of training and testing data as ordered sequences of OS event numbers.

(2) Normal traffic

Example 1:

XXX XXX XXX XXX XXX XXX XXX XXX ,

where XXX – any event that is not significant for reflecting attacks generated;

2. ***Examples of statistical attributes of particular sessions of a user performance on FTP-server level***

(1) R2L: FTP-crack

Example 1:

time	*duration*	*hot*	*failed_logins*	*logged_in*
17:53:45	1	0	3	0
17:53:45	1	0	3	0
17:53:46	0	0	2	0
17:53:47	1	0	3	0

(2) Normal traffic

Example 1:

time	*duration*	*hot*	*failed_logins*	*logged_in*
17:53:45	5	1	0	1
17:53:47	1	0	0	1
17:53:54	452	7	0	1

3. ***Examples of statistical attributes of FTP-server functioning during the short term time intervals (5 second)***

(1) R2L: FTP-crack

Example 1:

hot	*failed_logins*	*successful_logins*
0	12	0
0	14	0
0	6	0

(2) Normal traffic

Example 1:

hot	*failed_logins*	*successful_logins*
2	0	1
0	0	0
0	0	0
4	0	1

4. ***Examples of statistical attributes of FTP-server functioning during the long term time intervals corresponding to 10 – 60 minutes***

(1) R2L: FTP-crack

Example 1:

Hot	*failed_logins*	*successful_logins*
2	120	6
12	344	12
0	248	8

(2) Normal traffic

Example 1:

hot	*failed_logins*	*successful_logins*
1	1	8
8	0	15
15	2	11

4.5. Conclusion

In the Chapter the case study used for *development of software prototype of the IDLS components* is specified.

We determined *the categories and instances of attacks to be used in the case study*. To generate training and testing data we selected *four types of attack categories:* Probing; Remote to local (R2L); Denial of service (DOS); User to root (U2R). The exemplars of attacks selected for case study are SYN-scan, FTP-crack attack, SYN flood, and PipeUpAdmin.

The data sources used and *generic data structures* representing training and testing data of the selected sources are described. We have chosen three *data sources for training and testing data*: network-based (traffic level), host-based (operating system level) and application-based (FTP-server level). Each data source is represented by four *generic data structures*. These data structures correspond to the data produced on the basis of raw data processing. These data structure are as follows:

1. Time ordered sequence of values of binary vectors of parameters specifying significant events of raw data of a level (traffic level, OS logs and FTP-server logs);
2. Statistical attributes of particular connections (performance of a user) manifested in a data source (traffic level, OS logs and FTP-server logs);
3. Statistical attributes of traffic (users' activity) during the short term time intervals;
4. Statistical attributes of traffic (users' activity) during the long term time intervals.

The *instances of data structures representing training and testing data sets* to be used in data mining and KDD procedures are specified. The data structures of the network-based source (traffic level) are produced on the basis of processing of *tcpdump/windump* data. The data structures of host-based source (operating system level) were produced on the basis of processing of operating system log *Security* (for Windows 2000/XP). The data structures of application-based source (FTP-server level) were produced on the basis of processing of FTP-server log. We used for generating these data structures *TCPtrace* utility and several programs developed by the authors of the Report.

The *examples of training and testing instances* of each selected data source as they are used in learning procedures in case study are represented. These examples were presented for all five exemplars of attacks concerning to four attacks categories selected. We described also examples of data fixed for normal users' activity.

Chapter 5. Software Prototypes of Components of Multi-agent Intrusion Detection Learning System and Simulation Results

Abstract. This Chapter aims at simulation based assessment and validation of the main results of the Project that are methodology and technology for multi-agent IDLS design, implementation and deployment supported by the developed software tool and detailed realization of multi-agent architecture of generic IDLS. In this respect, the Chapter describes (1) the developed, implemented and deployed over a computer network components of the software prototype of a multi-agent Intrusion Detection Learning System destined for learning of intrusion detection in the framework of the elaborated case study (see Chapter 4) and (2) results of its testing. Design and implementation of the components of the above software prototype is accompanied by demonstration of the practical use of the developed methodology, technology and supporting software tool thus making it possible to validate them. The analysis of the testing results of the software prototype allow to conclude that the agent-based approach to IDL and elaborated methodology, technology and software tool constitute a promising starting platform for further research and development of the prospective IDLS. Detailed description of the training and testing procedures, intermediate and final results in all the steps of the IDS distributed learning including testing of particular base classifiers and also meta-classifier which produces the final decisions is given in *Appendix* following the Report itself.

5.1. Generic Architecture and Engineering of IDLS Software Prototype

Data Sources, Logical and Physical Hosts

Engineering of a multi-agent IDLS and its decision making component, IDS[1], starts with the preliminary analysis of the application domain, available data sources and their situation (location) within computer network. Data sources are here understood not as the sensor data but as records of databases, which are specified by their DNS names in ODBC managers of respective hosts.

In order to simplify the system configuration process, the notion of *data source logical host* is introduced. The logical host of a data source is determined by the IP address of the physical host and by the DNS name of the source's ODBC. Several logical hosts of data sources can be located on a single physical host of a LAN (Fig.5.1).

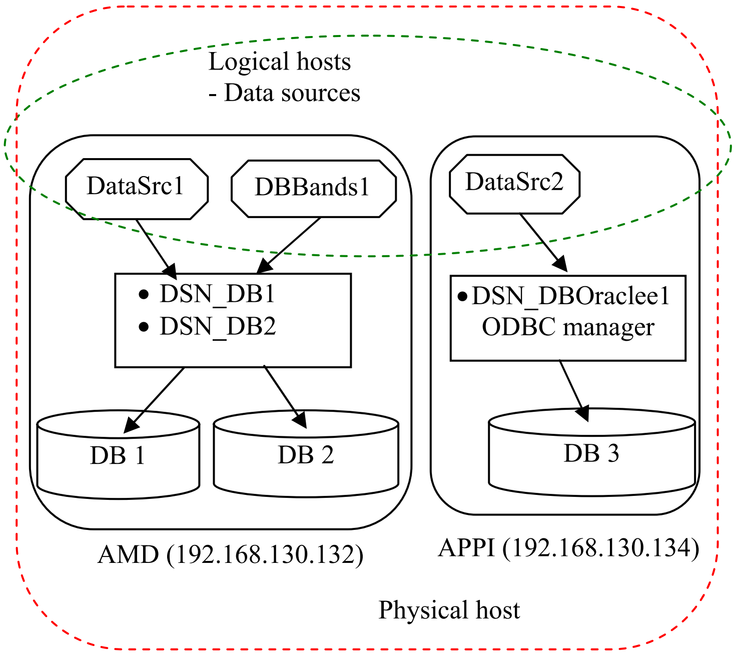


Fig.5.1. Explanation of notions of logical and physical hosts

Generation of IDL MAS components through Use of MASDK

According to the technology implemented to date (see Chapters 1, 2), in order to build an IDLS as applied Information Fusion MAS, the former is firstly specified in the *System Kernel* of the MASDK and resulting system is called "*Empty IDL MAS*". This specification includes making use of Generic agent (*invariant* component of any MAS including IDLS MAS) as a template for specification of agent classes (the agent classes are the same for different IF applications and that is why they can be specified in advance and afterwards used as reusable components in different IF systems, including

[1] We suppose further that term "*IDLS*" and "*IDL MAS"* concerns to the multi-agent that solves both intrusion detection learning and intrusion detection tasks.

IDL system). Problem domain ontology (see Fig.1.8 and also section 2.5) can be also specified one time by use of Ontology Editor of MASDK and then used in different IF applications including IDL one. Next step of IDL MAS specification consists in specification of the agent classes, that is specification of the agent applied component (it is different for each agent class, see Fig.1.8). At the same time the shared component of application ontology is specified by use of a special editor that is now a component of Information Fusion Learning Toolkit (see section 2.3). Afterwards agent classes are replicated into agent classes instances and installed in predefined computers of a LAN within communication environment previously deployed (at any time) within this LAN (remind that communication environment consists of portals installed and run in each computer in which agents of IDLS should be deployed and server which can be installed in any computer of LAN, see section 3.1 of this Report). The resulting multi-agent IDLS can be called as "empty system" which has to be further "filled in" by particular content that is data and knowledge interpreting particular ontology notions, providing particular agent procedures (for example, state machines, learning and decision making components of the software) with concrete data. Most of the above operations are done use of MASDK software editors in accordance with the technology thoroughly described in Chapters 1 and 2 of this Report.

Thus, the initial specification of an applied IDL MAS in the System Kernel of MASDK contains specification of the following classes of agents (see Fig.5.2 and also Chapter 3 explaining destinations of the below agent classes):

- *Data Source Manager (DSM)* – class of agents that support the data source interface;
- *BC* – class of agents that solve the task of the input data's basic classification;
- *MC* – class of agents that enable the combination of base classifiers' decisions;
- *KDD* – class of agents responsible for the training of base and meta–classifiers. Functionalities of agents of this class combine the functionalities of *KDD agent of data source* and *KDD meta–agent*;
- *KDD Master* – class of agents that enable the design and specification of the Decision Fusion meta–model and manages of the base and meta–classifiers training and testing.

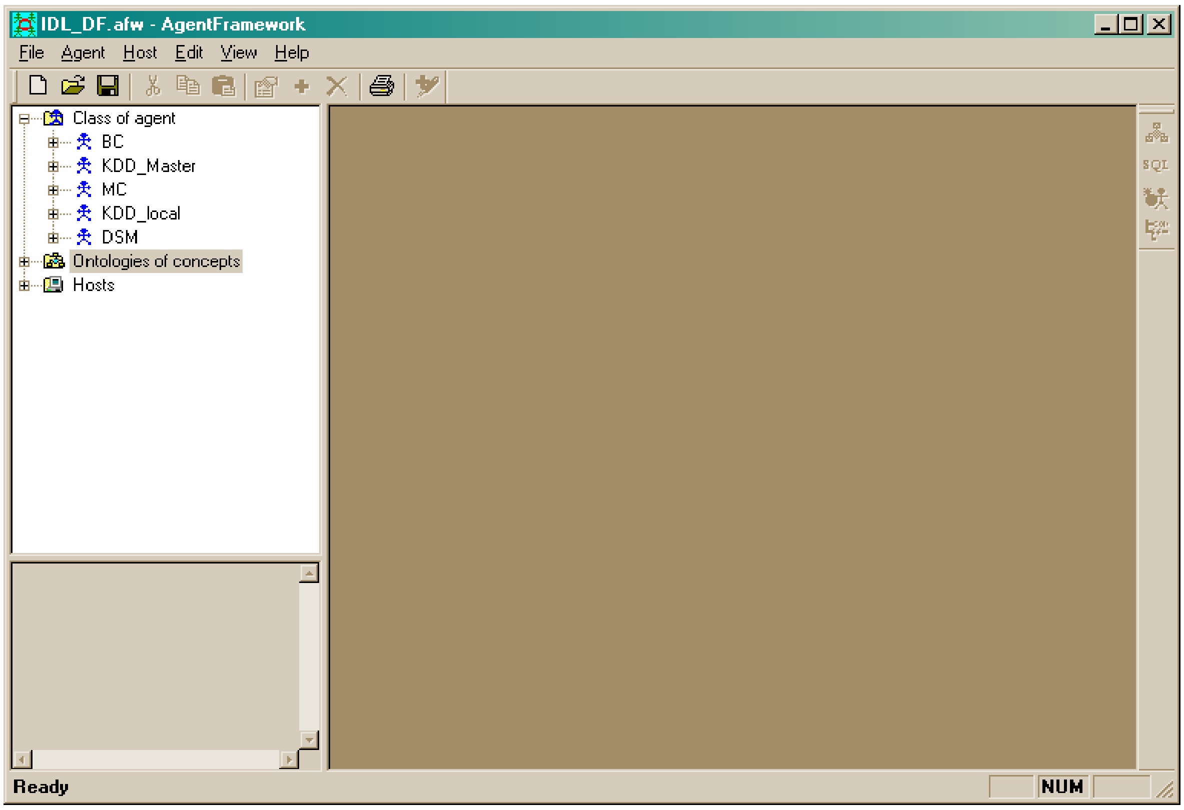
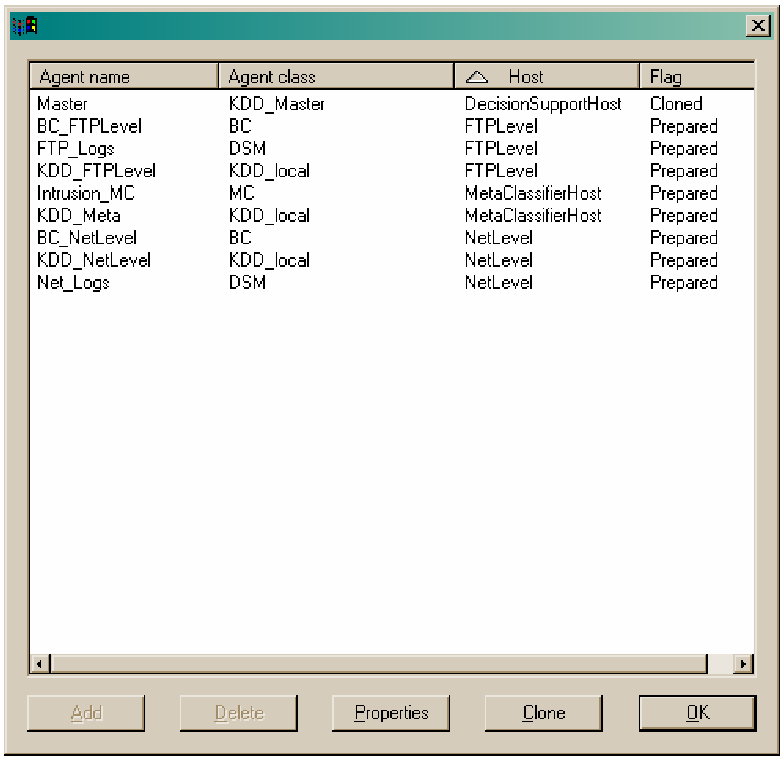


Fig.5.2. User interface for initial specification of the IDLS agent classes in System Kernel of MASDK

As it was above pointed out, the process of specification of the basic configuration of the applied MAS in *System kernel* implies:

- o replication of the instances of the corresponding agent classes, and
- o designation of *logical* and *physical* hosts that determine the "space of existence" for instances of software agents within given computer network (see Fig.5.3).

The use of the notion of *logical host* is necessitated by technical reasons. Relationships are defined between physical and logical hosts, according to which, one or more logical hosts correspond to each physical host of LAN. The use of the notion of logical hosts allows for specifying the location of



Agent name	Agent class	Host	Flag
Master	KDD_Master	DecisionSupportHost	Cloned
BC_FTPLevel	BC	FTPLevel	Prepared
FTP_Logs	DSM	FTPLevel	Prepared
KDD_FTPLevel	KDD_local	FTPLevel	Prepared
Intrusion_MC	MC	MetaClassifierHost	Prepared
KDD_Meta	KDD_local	MetaClassifierHost	Prepared
BC_NetLevel	BC	NetLevel	Prepared
KDD_NetLevel	KDD_local	NetLevel	Prepared
Net_Logs	DSM	NetLevel	Prepared

Fig.5.3. Specification of IDL MAS configuration

software agents in physical hosts in the IDL MAS specification given in the *System Kernel*.

Dialog window of MASDK shown in Fig.5.3 is used for specification of configurations of software agents of the corresponding classes, including the indication of their location in terms of logical hosts.

In accordance with the given data sources, the configuration of the software agents' instances of the IDL MAS software prototype under development is formed in the following manner:

1. For each data source, one logical host and three instances (according to the number of data sources, see Chapter 4) of software agents *DSM, BC, KDD agent* are specified.
2. To specify the meta–level component of IDL MAS (let us remind that meta-level component is responsible for combining decisions of base classifiers and IDL MAS in design and operation modes management), one or several logical hosts are specified. In each logical host, one instance of software agents *MC* and *KDD local* is specified.
3. Agents of class *KDD Master* that support the management of training and decision making processes should be located on the same logical host.

Thus, the hosts of three types may be defined in the system with the agents' allocation to them as it is shown in Tab.5.1. In Fig.5.4 the configuration of the IDL MAS prototype under development specified by use of MASDK is presented within the respective dialog window of the MASDK user interface.

Generation of software agents is preceded by the stage at which correspondence between physical and logical hosts is established. This is done through a dialog window of MASDK shown in Fig.5.5. Specification of instances of software agents of the *DSM class* is expanded at the generation stage with the following data:

Table 5.1. Allocation of the agents on hosts

Type of system host	Necessary agent of host
Local data source	DSM agent BC agent KDD local agent
Meta-classification host	MC agent KDD local agent
Meta-training host	KDD Master agent

- Name (identifier) of data source that the instance of agent “works with”. This name is further used as identifier of the source;
- Name (identifier) of the instance of base classifier training agent that “works” with that data source;
- Name (identifier) of the instance of classification agent that is tuned for working with the aforementioned data source;
- Name (identifier) of connection to database viewed as the source in the ODBC manager.

The procedure for replication (cloning) of the mentioned instances of software agents of given classes is conducted through a dialog window shown in Fig.5.3. To that time, an auxiliary program Portal (see Chapter 3) must be launched on all physical hosts of the network during the generation of

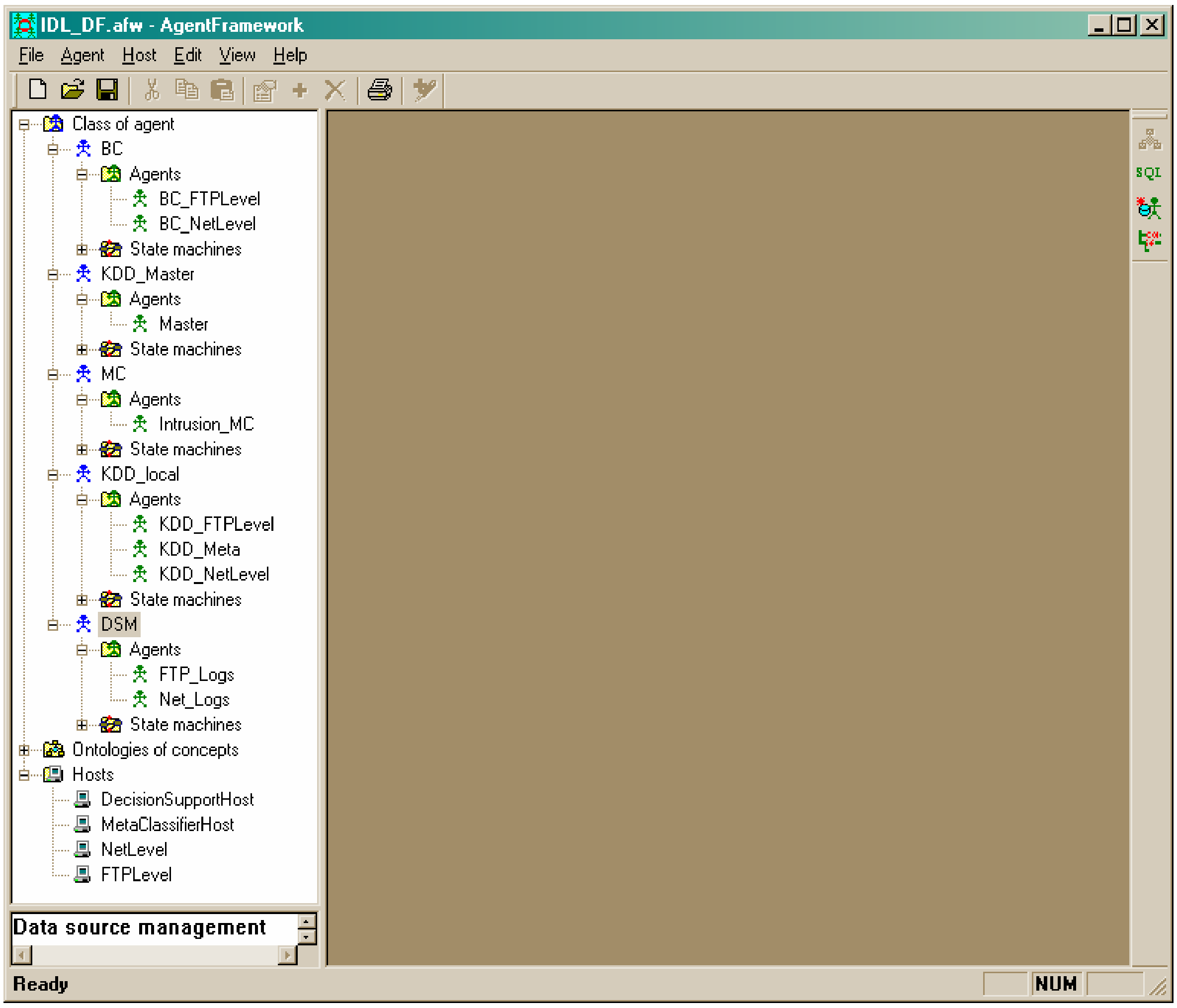
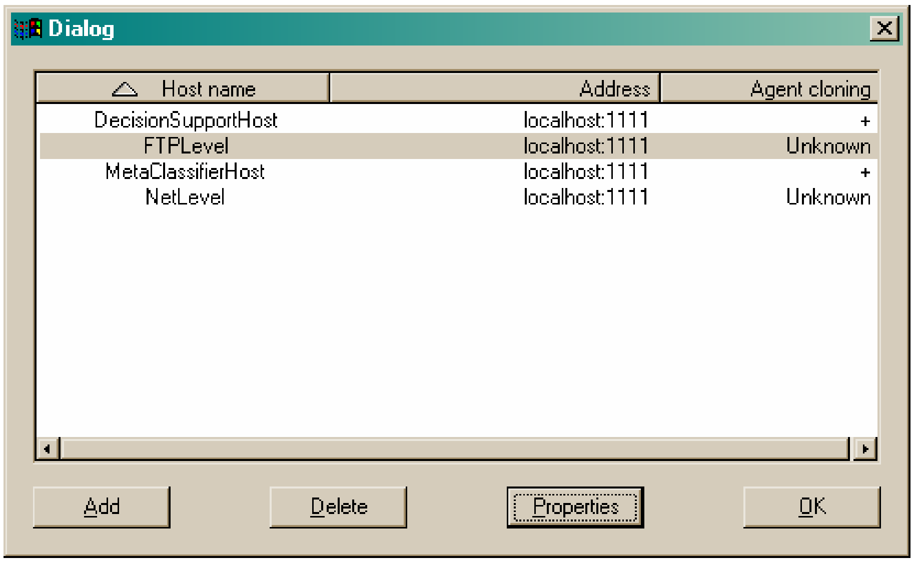


Fig.5.4. Specification of the system configuration in MASDK



Host name	Address	Agent cloning
DecisionSupportHost	localhost:1111	+
FTPLevel	localhost:1111	Unknown
MetaClassifierHost	localhost:1111	+
NetLevel	localhost:1111	Unknown

Fig.5.5. Dialog for establishing correspondence between physical and logical hosts

agents.

After the generation of the instances of the agent classes, the IDL MAS operates in the environment autonomously, i.e. independently of MASDK. In the learning mode, training and testing of IDS classification and decision combining agents is carried out.

For the system's operation, Portals should be run on all the system physical hosts, and the IDL MAS agents must be launched by the command of IDL MAS server.

To start the system's operation in learning mode (this assumes the meta-learning agent started), the command *Start system* should be chosen from the list of the predefined user commands and interfaces of that agent. At the system's first start, after its deployment, an additional initialization protocol will be launched automatically, which configures the system in accordance with the specification of the existing local data sources and the system's hosts. All the information about the system's configuration is distributed among the respective agents of the system, and agents switch into operational mode, fully ready to fulfill the system's functions.

The subsequent engineering operations are supported by Information Fusion Learning Toolkit (see section 2.3).

5.2. Intrusion detection KDD Master Agent

5.2.1. Meta-level Ontology Editing

The process of creating the target system starts with the *design of the application ontology*. In the current prototype of the system, the most simple of the developed protocols for the distributed application ontology engineering is implemented ("top-down" operation; this protocol is specified formally). Conceptually, this protocol consists of several steps. First, the shared application ontology is designed by the meta-level designer ("manager of meta–level") with the subsequent dialog with other (distributed) designers via mediation by *KDD Master Agent*. The application ontology designed by manager of meta–level is forwarded to the instances of the agents of the class *Data Source Manager (DSM)*, which enable the interface with particular data sources. Then, the sources' experts (designers working with the particular data sources) tune the interpretation functions of the corresponding fragments of the ontology to the data sources databases.

To create the shared application ontology, the ontology editor implemented in the *KDD Master agent*, is used. The editor is destined for solving the following subtasks:

- Designing and editing the shared application ontology. As a result, a specification of the notions of the ontology, their attributes, and corresponding value domains are formed.
- Specification of the data of the local data sources in terms of the shared ontology. As a result of this task solving for each local data source in the shared application ontology, the fragment of it

is specified that further will be used in IDS operation with the data of that source.

- Creating and editing the ontology notions derived (secondary) properties. Solving this task results in the creation of the list of secondary properties of the notions specified in terms of their attributes.
- Creating and editing of the list of data classes. Solving this task results in forming the list of classes used for the specification of the input data interpretations in terms of the labels of classes of statuses of connections (*Normal*, *Abnormal*, etc. if any).

Before use of the ontology editor, the *KDD Master* agent must be activated. The ontology editor is called from that agent through activating the command *Ontology editor* in the user interface.

The ontology designed is an aggregate of notions with attributes. The entities may be grouped into equivalence classes (they are called *structure classes*). To determine the type of attribute, the predefined domain “Type of attribute” is used. It contains the possible values of types of attributes acceptable in the system.

The values of this domain are taken from the *PMML* (Predictive Model Markup Language) standard. Their explanation is given in Table 5.2.

Table 5.2. Values of domain “Type of attribute of the ontology”

Value of domain	Attribute’s properties
bool	Boolean scale (2 possible values: True/False)
categorical	Categorical scale – set of predefined unordered values
ordinal	Ordered scale – set of predefined values with an order relationship imposed on them
continuous	Numerical scale

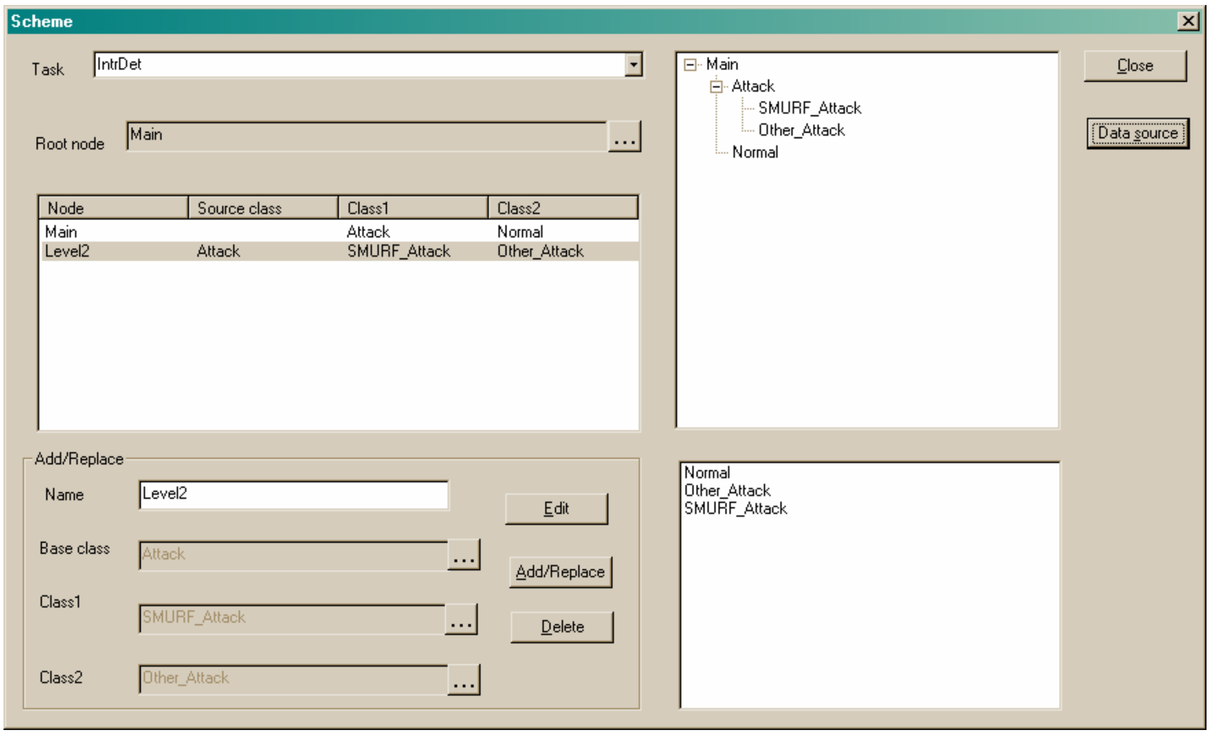
5.2.2. Editing of the Decision Fusion Meta–Model

The structure of distributed learning and classification used in IDL MAS, *Decision Fusion meta–model* is used (see Chapter 1). The latter is specified by exploiting an aggregate of user interfaces implemented in agents of class *KDD Master*. The respective editor that is a component of Information Fusion Learning Toolkit (see section 2.3) is activated by the command *Decision making editor*.

The main window of the editor shows the list of names of classification tasks already created by user. To specify and add the name of a new task, the fields *Name*, *Description* and *Base entity* are used. Arbitrary name of the task is entered in the *Name* field. In the *Description* field, a detailed comment for the task is entered. In the field *Base entity,* the notion of the application ontology is chosen, for which the classes have been determined and specified for the classification of instances of that notion. The notion chosen as the subject of classification will hereinafter be called “*Base entity*”. For one base entity, several classification tasks may be defined in the task list; their list and hierarchy are specified by the *classification tree* (see Fig.5.6). The field *Entity description* contains comments for the chosen entity of the ontology formed in the description of the ontology.

For detailed description of the classification task chosen from the name list and the classification tree, the editor is used, which is activated by the *Scheme* button. The user window of this editor is shown in Fig.5.6. In the description of the classification task and the decision making scheme, the set of classes specified for the base entity is used. In particular, in this example four classes have been specified: *Attack*, *Normal*, *Smurf attack* and *Other attack*. The meaning of this particular classification task lies in assigning a class of decision that corresponds to one of the leaves of the classification tree to each instance of the base entity.

A group of fields unified by the function *Add/Replace* is used to specify a new node or to edit an old node. The name of the node of classification tree is edited in the *Name* field. Fields *Class 1* and *Class 2* show names of classes of decisions that can be made in that node (the decision corresponds to one of the two branches of the tree stemming from it). The field *Base class* shows the name of the class assigned to the base entity at the previous level of the classification tree. For example, in the tree


Scheme

Task: IntrDet

Root node: Main

Node	Source class	Class1	Class2
Main		Attack	Normal
Level2	Attack	SMURF_Attack	Other_Attack

Add/Replace

Name: Level2

Base class: Attack

Class1: SMURF_Attack

Class2: Other_Attack

```
Main
  Attack
    SMURF_Attack
    Other_Attack
  Normal
```

```
Normal
Other_Attack
SMURF_Attack
```

Fig.5.6. Editor of the classification tree of the Decision Fusion meta-model

shown in Fig.5.6, the role of *Base class* for the node *Level2* is played by the class *Attack*. The name set in correspondence to the variable *Base class* for the current node is also called "precondition class".

Thus created decision tree is shown in the top right field. The bottom right field shows the set of leaves of the created tree that corresponds to the set of possible solutions specified during the classification of the instances of base entity.

For a single classification task, several classification schemes can be formed, each having a corresponding classification tree. The identification of classification schemes is done based on the name of root node of the corresponding classification tree. The example (Fig.5.6) shows one classification tree with the root node called *Main*. In case several classification trees have been formed, the top right window shows the structure of the decision making tree whose root node's name is selected in the field *Root node*.

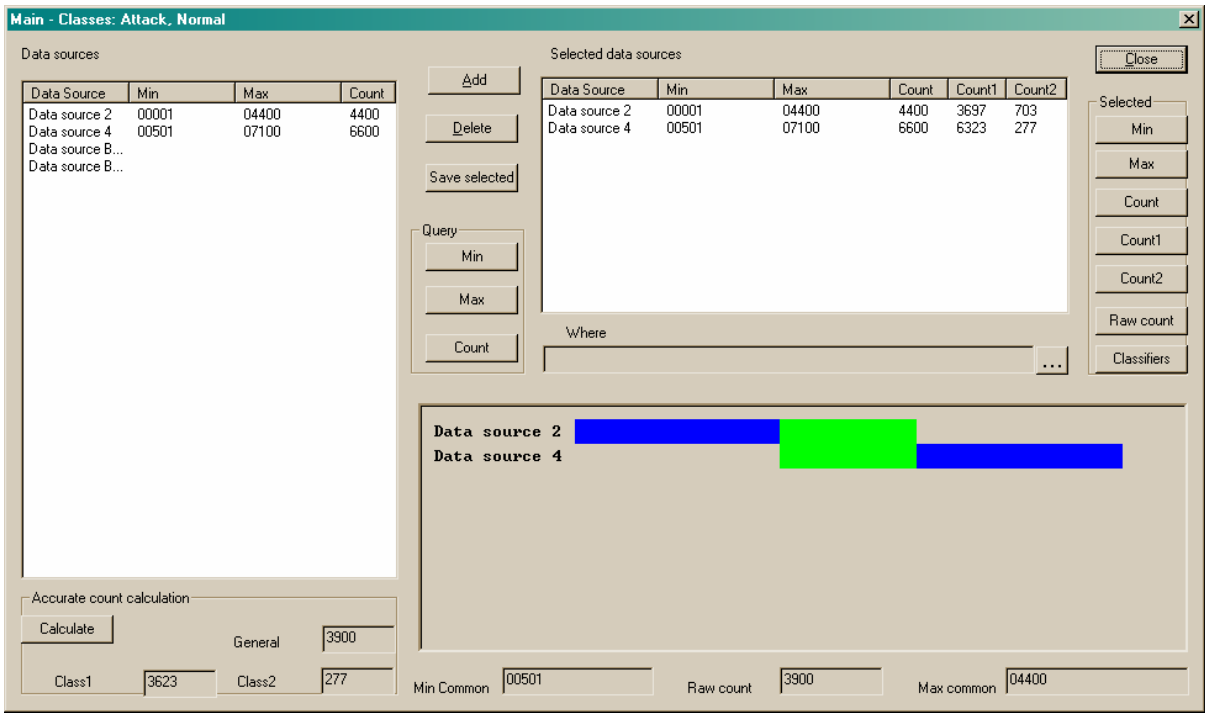
Let us remind that names of three classes, *Class 1*, *Class 2* and *Base class,* are set in correspondence to each node of classification tree. In creating classification trees, the editor oversees the meeting of two conditions (constraints): (1) all three names have been specified for one intermediate node of classification tree, and those names are different; (2) for one decision making task, there can be no two different nodes with the same class of precondition.

5.2.3. Analysis of Data Available for Classifiers Training and Testing

Classification tree considered in the previous subsection consists of a number of nodes. Each of them corresponds to a certain decision making scheme, which includes several base classifiers and at least one meta–classifier. This scheme is called Decision Fusion (DF) meta–model (see section 1.6.2 for details). The creation of DF–meta–model requires preliminary analysis of the input data recorded in the databases of sources[1]. The process of data analysis is initiated by the button *Data Source* in the dialog window shown in Fig.5.6. The data analysis process will take place for the classifiers involved in the decision-making process in the current (under consideration) node of the classification tree.

Analysis of possibilities for choice of data: node of decision-making tree

[1] Peculiarities of the methodology of allocation of training and testing samples to particular base- and meta–classifiers were discussed in section 1.7)



Main - Classes: Attack, Normal

Data sources

Data Source	Min	Max	Count
Data source 2	00001	04400	4400
Data source 4	00501	07100	6600
Data source B...			
Data source B...			

Selected data sources

Data Source	Min	Max	Count	Count1	Count2
Data source 2	00001	04400	4400	3697	703
Data source 4	00501	07100	6600	6323	277

Accurate count calculation

General 3900

Class1 3623 Class2 277

Min Common 00501 Raw count 3900 Max common 04400

Fig.5.7. User interface for analysis of data available for training and testing of the classifiers corresponding to the chosen node of classification tree

The analysis of data by use of which the classifiers of the chosen node of classification tree will be trained and tested is performed through the dialog shown in Fig.5.7.

The table "*Data Source*" shows the list of available data sources. For each of them, the following parameters are specified: *Field Min* – the minimal value of the identifier of the instance of base entity (key) recorded in the database of data source; *Field Max* – the maximum value of the identifier of the instance of base entity (key) recorded in the database of data source; *Field Count* – the total number of instances of base entity recorded in the database of data source.

These values are calculated through the buttons of the same names from the group of buttons *Query*. Results of queries are stored in the *Intrusion Detection KDD Master agent's* knowledge base and appear in the table during further operation.

The table "*Selected data sources*" shows the list of data sources, with which that node "works". As a rule, this list comprises all the sources whose databases contain the specifications of instances of the base entity. For each of them, in addition to the data already calculated, the following characteristics can be obtained:

- Number of instances of the base entity, for which the first class from the description of the chosen node of classification tree (field Count 1) is set in correspondence as the result of classification;
- Number of instances of the base entity, for which the second class from the description of the chosen node of classification tree (field Count 2) is set in correspondence as the result of classification.

Queries to obtain these characteristics are formed through the buttons of the same names from the group of buttons "*Selected*".

For the data sources chosen by user, the diagram is automatically created (bottom right window) that represents a "*raw top count*" of the number of common instances having interpretations of base entity in the databases of all data sources. In the diagram, sets of common instances of base entity are shown in green, and the value of that estimate is shown in field *Raw count* in the lower part of the window. Calculation of this interval is performed at the click on the button *Raw count*. This estimate is calculated as the minimal number of common interpretations on all chosen data sources within the common interval between the minimal value of the identifier in the field *Max* and the maximum value on the field *Min*. These values are shown in fields *Min common* and *Max common* in the lower part of dialog window.

The exact number of common instances of base entity is shown in the group of fields "*Accurate count calculation*" in the field *General.* Fields *Class1* and *Class2* show the number of common instances of base entity interpreted respectively as the first and the second class of possible decisions corresponding to the node of decision-making tree under consideration. These values can be calculated after the complete lists of identifiers of recorded instances of base entity from each of the local data sources have been obtained. Query to obtain those lists is initiated at the click on the button *Accurate count calculation* and is performed with consideration to the classes of decisions.

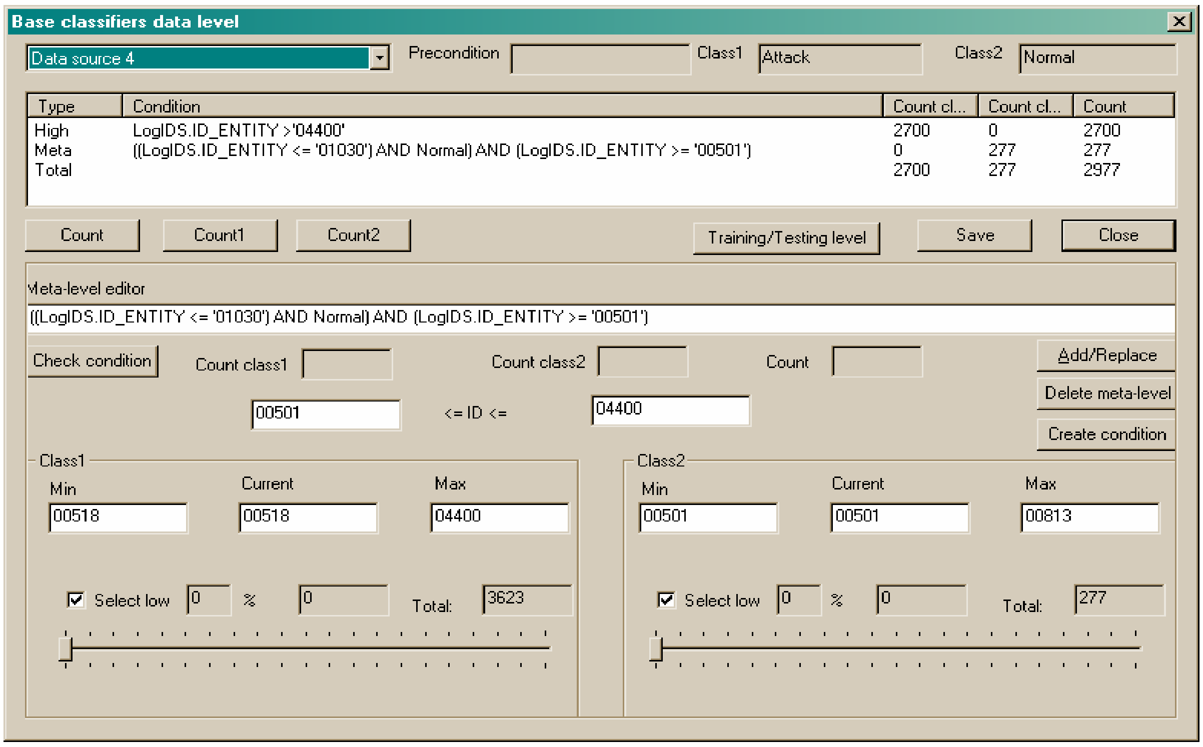
The number of instances that belong to the same entity in different data sources is determined in order to identify fragments of data that can be used for the training of meta– and/or base classifiers. Here, all instances of the base entity can be used for the training and testing of base classifiers. The scenario for putting together the list of such instances is described in the next section.

Choice of data: node of classification tree, basic classification

The choice of data for training and testing the base classifiers of a single data source starts with the dialog shown in Fig.5.8. The dialog is opened by clicking on the button *Classifiers* in the dialog shown in Fig.5.7. The name of the chosen data source is shown in the top part of the window.

The top part of the window shows particular ranges of values of keys of instances recorded only on the corresponding data sources (the set of such instances is highlighted blue in Fig.5.7). These ranges can be of two types: less and/or more than range of instances common to all data sources. To describe the obtained separate ranges of instances and to further work with them, a criterion for their selection is formed automatically. These ranges of instances will be further used for the training and testing of base classifiers. Ranges of instances common to all data sources will mainly be used for training and testing of meta–classifiers; however, they can also be used for training and testing of base classifiers. In the last line of the table marked by the label *Total*, the total number of instances of entities is shown for designer's information.

The data for the training of base classifiers from the range of instances common to all data sources is described in the group of fields corresponding to *Meta-level editor.* In this part of the editor, user can form, in manual or semi-automatic mode, the condition of selection for this data, and the boundaries of the selected range of events, which are shown in the fields that are marked with label *ID.*



Base classifiers data level

Data source 4 | Precondition | Class1 Attack | Class2 Normal

Type	Condition	Count cl...	Count cl...	Count
High	LogIDS.ID_ENTITY >'04400'	2700	0	2700
Meta	((LogIDS.ID_ENTITY <= '01030') AND Normal) AND (LogIDS.ID_ENTITY >= '00501')	0	277	277
Total		2700	277	2977

Meta-level editor

((LogIDS.ID_ENTITY <= '01030') AND Normal) AND (LogIDS.ID_ENTITY >= '00501')

00501 <= ID <= 04400

	Min	Current	Max	Select low	%		Total
Class1	00518	00518	04400	✓	0	0	3623
Class2	00501	00501	00813	✓	0	0	277

Fig.5.8. Data selection for training and testing of base classifiers

By clicking the button *Check condition,* the number of instances for each class of the node and the node's total number of instances in accordance with the condition introduced by user, is calculated.

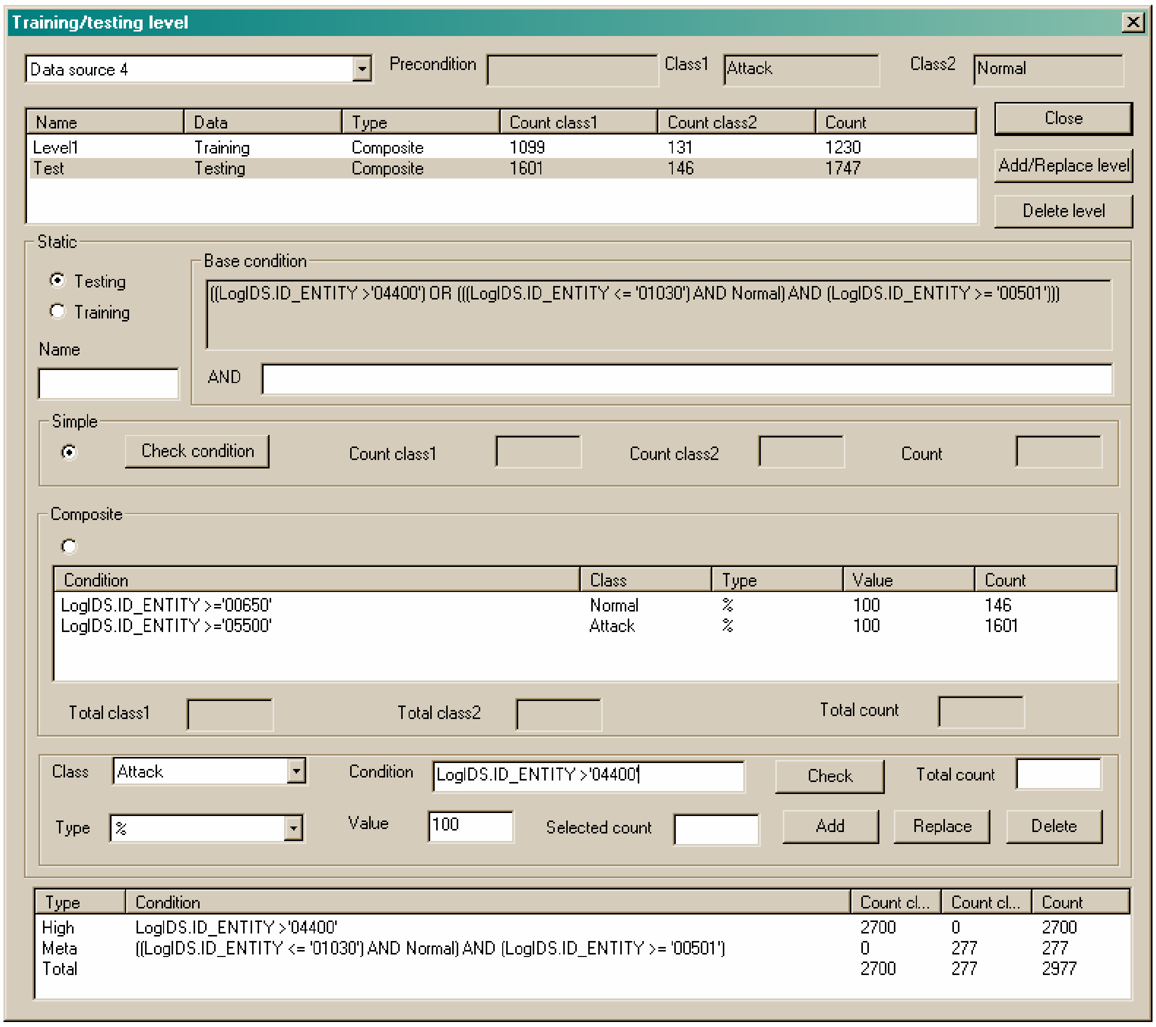
To create the selection condition in semi-automatic mode, the button *Create Condition* and groups of fields for each of the target classes are used. Each of the groups of fields *Class 1* and *Class 2* shows the maximum (field *Max*) and minimal (field *Min*) value of identifiers, as well as the total number of instances of the corresponding class. Here, the location of the marker on the scale corresponds to the percentage of the chosen interpretations of the corresponding class of their total number. The percentage, the value of identifier that corresponds to that percentage and the exact number of selected instances are shown in the respective fields above the selection scale.

After the selection of data used for the training and testing of base classifiers, they need to be divided into training and testing. This procedure takes place in a particular dialog initiated by clicking on the button *Training / Testing level*. Here, the editor opens as shown in Fig.5.9.

The training and testing data are shown as descriptions of groups of data used for training and testing of the respective base classifiers.

Each group of data can be of simple type (all selection conditions are specified by one condition) or composite type (data is formed from several subsets, each one specified by its own selection condition). For each of these types, the dialog provides a group of elements of interface. The type of the sample is changed by the *Simple / Composite* toggle.

Regardless of the type, there are common selection conditions for data groups. They consist of two parts. The first part is formed automatically on the basis of all existing group of training and testing



Training/testing level

Data source 4 | Precondition: | Class1: Attack | Class2: Normal

Name	Data	Type	Count class1	Count class2	Count
Level1	Training	Composite	1099	131	1230
Test	Testing	Composite	1601	146	1747

Static: [✓ Testing] Training

Name:

Base condition: ((LogIDS.ID_ENTITY >'04400') OR (((LogIDS.ID_ENTITY <= '01030') AND Normal) AND (LogIDS.ID_ENTITY >= '00501')))

AND

[✓ Simple] — Count class1: | Count class2: | Count:

Composite

Condition	Class	Type	Value	Count
LogIDS.ID_ENTITY >='00650'	Normal	%	100	146
LogIDS.ID_ENTITY >='05500'	Attack	%	100	1601

Total class1: | Total class2: | Total count:

Class: Attack | Condition: LogIDS.ID_ENTITY >'04400' | Total count:

Type: % | Value: 100 | Selected count:

Type	Condition	Count cl...	Count cl...	Count
High	LogIDS.ID_ENTITY >'04400'	2700	0	2700
Meta	((LogIDS.ID_ENTITY <= '01030') AND Normal) AND (LogIDS.ID_ENTITY >= '00501')	0	277	277
Total		2700	277	2977

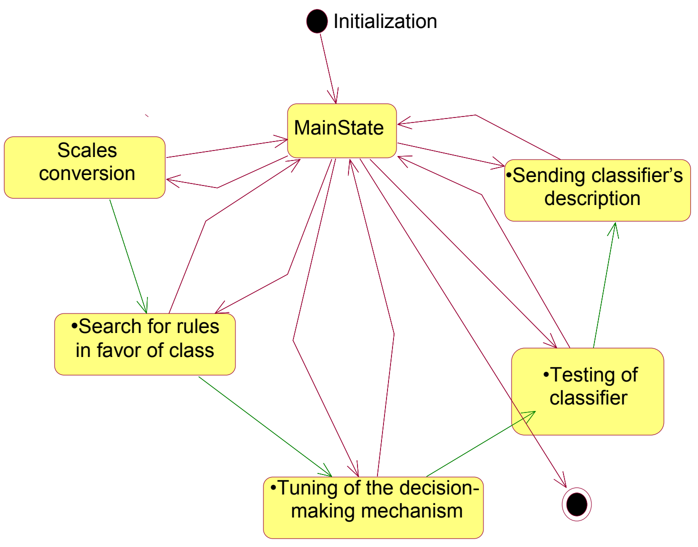
Fig.5.9. Data selection for training and testing of a classifier

data for base classifiers (conditions of all levels of basic classifications are united through the logical connective "*OR*"). The second part is formed by user in the interface and is always joined with the first one by the logical connective "*AND*".

If the sample is of simple type then this condition is the only one, and it determines the selected subset of data. The number of instances of classes and the total number of instances of the sample may be obtained through sending a query to the data sources. For that, the button *Check condition* is used. The data received from the local data source is stored in the fields *Count, Count Class1*, *Count Class2*.

For composite group of data, besides the common condition, particular selection conditions are specified for each group of data. They are recorded in the list in the group of fields *Composite.*

The selection conditions include the name of the class and the percentage of the records from the total number of records, or from their explicit number. Each condition is formed in the group of fields in the lower part of the dialog.


Initialization
MainState
Scales conversion
•Sending classifier's description
•Search for rules in favor of class
•Testing of classifier
•Tuning of the decision-making mechanism
Fig.5.10. Base classifier training scenario

The table and list are updated by clicking the *Add/Replace* button. The condition is checked by clicking on the *Check* button.

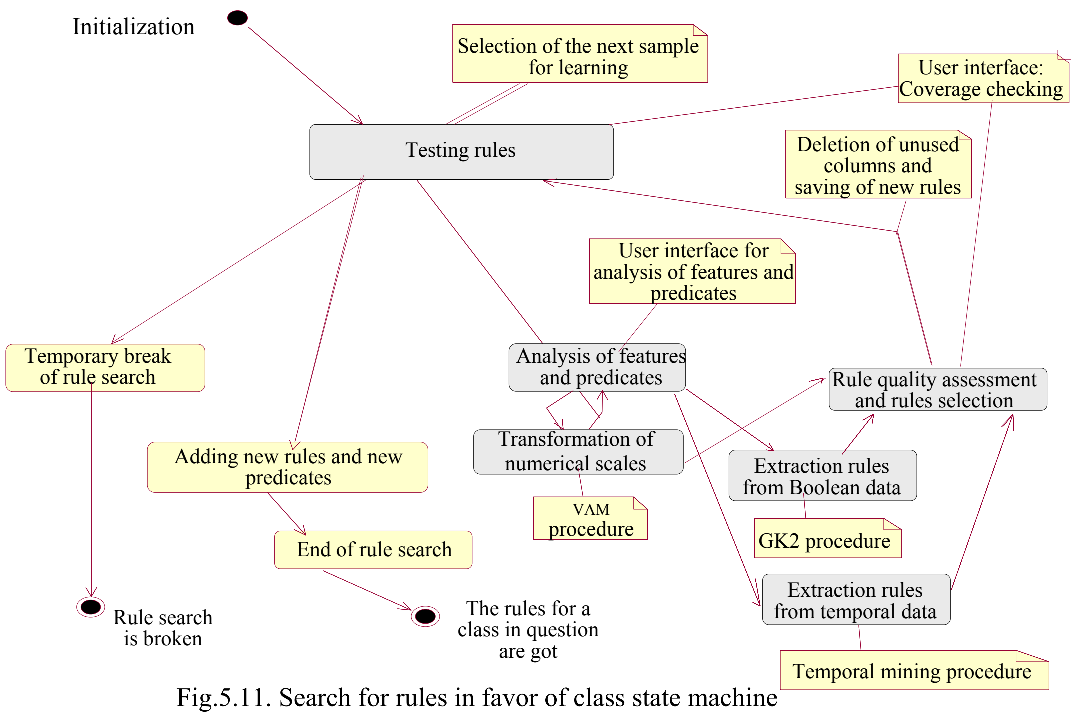
Here, a query is sent to the data source, and the results are shown in the fields *Selected Count* and *Total count*.

5.3. Intrusion Detection KDD–Agent of a Source

5.3.1. Base Classifiers Training Scenario

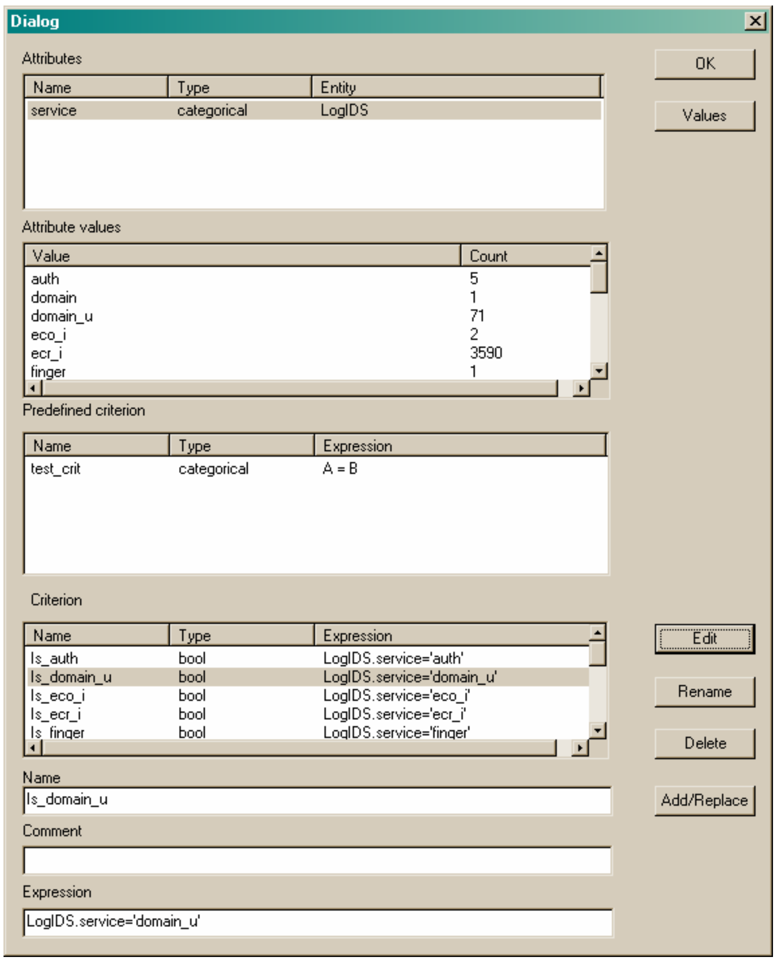
The base classifier training scenario consists of a number of particular subtasks performed in a certain order. This scenario is represented by the state machine shown in Fig.5.10. In each of its states, the user solves particular subtasks through the appropriate dialogs. Let us consider these states:

- *Main State*. In this state, the dialog interface is shown, through which the training of a single base classifier is managed and monitored.
- *Scales conversion*. This subtask is considered in the case if the training data contains the attributes of the *ordinal* or *categorical* type. These attributes should be converted to the scales, for which implemented algorithms exist in the *Library of training and testing methods*.
- *Search for rules in favor of class*. In this state, the main training subtask is solved that involves search for rules based on the analysis of the training dataset. The search for rules may take place several times, supplementing the set of previously found rules. The system behavior scenario in this state is implemented through a nested state machine shown in Fig.5.11.
- *Tuning of the decision-making mechanism*. It is assumed that in this state, all the decision-making rules are found for the classifier, and the general decision making mechanism that combines decisions made by separate rules, is being tuned (specified).
- *Testing of classifier*. In this state, the classifier is tested and its performance is analyzed.
- *Sending classifier's description.* In this state, all rules and parameters of the decision-making mechanism are sent to the agent of base classifier.



Initialization
Selection of the next sample for learning
User interface: Coverage checking
Testing rules
Deletion of unused columns and saving of new rules
User interface for analysis of features and predicates
Temporary break of rule search
Analysis of features and predicates
Rule quality assessment and rules selection
Adding new rules and new predicates
Transformation of numerical scales
Extraction rules from Boolean data
VAM procedure
GK2 procedure
End of rule search
Extraction rules from temporal data
Rule search is broken
The rules for a class in question are got
Temporal mining procedure

Fig.5.11. Search for rules in favor of class state machine



Dialog

Attributes

Name	Type	Entity
service	categorical	LoglDS

Attribute values

Value	Count
auth	5
domain	1
domain_u	71
eco_i	2
ecr_i	3590
finger	1

Predefined criterion

Name	Type	Expression
test_crit	categorical	A = B

Criterion

Name	Type	Expression
Is_auth	bool	LoglDS.service='auth'
Is_domain_u	bool	LoglDS.service='domain_u'
Is_eco_i	bool	LoglDS.service='eco_i'
Is_ecr_i	bool	LoglDS.service='ecr_i'
Is_finger	bool	LoglDS.service='finger'

Name: Is_domain_u

Comment:

Expression: LoglDS.service='domain_u'

Fig.5.12. Editor of characteristics conversion

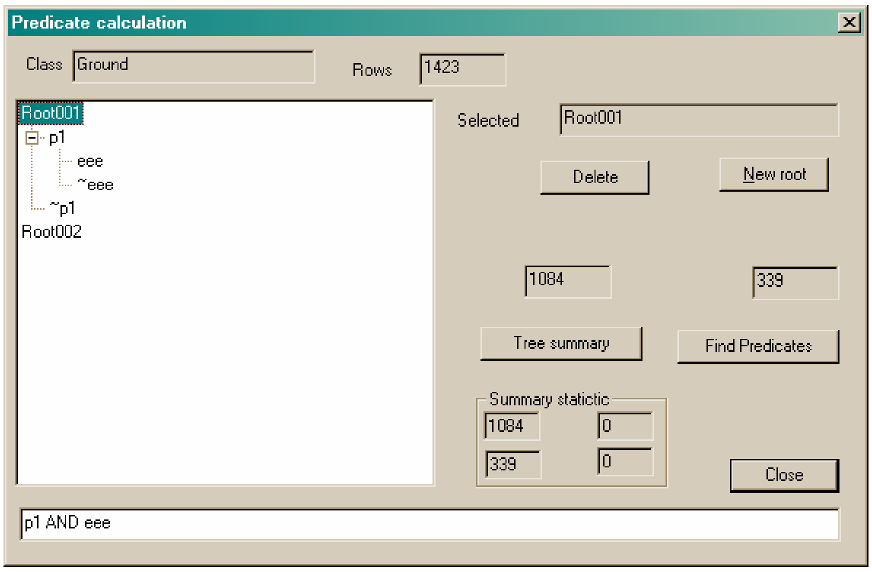
5.3.2. Conversion of Features

Current task is solved in the *Scales conversion* state. Conversion of the classification features is carried out through the interface shown in Fig.5.12. This task is aimed at converting the features represented in categorical and ordinal scales into numerical or Boolean. The conversion is done by user through the creation of new features and setting calculation functions for them that will use the existing and previously entered features and attributes as arguments.

The user editor window shows the specification of the initial features and attributes, and that of the ones added by user. The initial features and attributes are shown in different lists (the initial features belong to the shared ontology of the application domain, and will never be changed by user in the training process). For each of categorical and ordinal features and attributes on the training sample data, a list of values with the number of their respective instances can be drawn. To obtain such list of values by a selected attribute, click the *Values* button.

The functions for the calculation of the new characteristics are described as predicates with arithmetic terms and functional symbols (“logical-arithmetical expressions”) that use the initial attributes, features and their values as terms. In the process of creating these expressions, there is an option of inserting the existing attributes, features and their values from the lists into the expression line.

5.3.3. The VAM Method


Predicate calculation

Class	Ground
Rows	1423
Selected	Root001

```
Root001
  p1
    eee
    ~eee
  ~p1
Root002
```

1084	339

Summary statictic

1084	0
339	0

p1 AND eee

Fig.5.13. Main window of the VAM component

The VAM (*Visual Analytic Mining*) module is designed for extracting production rules[1] from the training sample represented in real scales (See [InterRep#2] for details). Each of these rules can be viewed as a pattern, on whose basis other rules and patterns can be formed. Then the obtained patterns, in particular, are used together with binary features for the creation of the classifiers’ rules base.

The main window of the component that implements the *VAM* method is shown in Fig.5.13. Left part of the window shows the structure of the separation rules. This structure is a set of binary “trees” where each node corresponds to one separation rule (predicate) defined over numerical features. Each tree as a whole describes one production rule represented symbolically in the lower part of the screen.

The search for the predicate that corresponds to the node of the tree is conducted on the basis of a subset of data selected through the predicates of the higher nodes of the tree.

The top part of the dialog shows information about the class, for which the production rules are being formed, and information about the size of data in the training sample. At the beginning of the training, there is only one tree with a single node that is tree’s root. In the process of training the user may create a new tree, or continue training for the existing leaf of the tree. For the selected leaf, the information about the number of instances of the class and the “counter-class” is shown. Clicking on the *Find Predicates* button will start the procedure of searching for predicate that corresponds to the selected leaf of the decision tree. After that, the user interface for searching and design predicates (Fig.5.14) opens for the subset of data that corresponds to the selected node.

The search for predicate consists of two separate subtasks:

- Search for “informative subspaces” in which the data of the class are best separated from the rest of the data, and
- Creating the separation line in the selected subspace.

The procedure of searching for informative subspaces may take place with different search parameters specified. Each search procedure is documented as particular search task. The resulting task list is reflected in two lists:

- *Calculated sequence* – list of tasks for which the search procedure has been completed;
- *Non-calculated sequence* – list of tasks that have not been completely solved (the procedure of searching for informative subspaces on a large sample in a space with a high dimension may take a long CPU time; therefore, it is implemented in such a way that it can be paused and the

[1] Practically, it extracts premises for given conclusion.

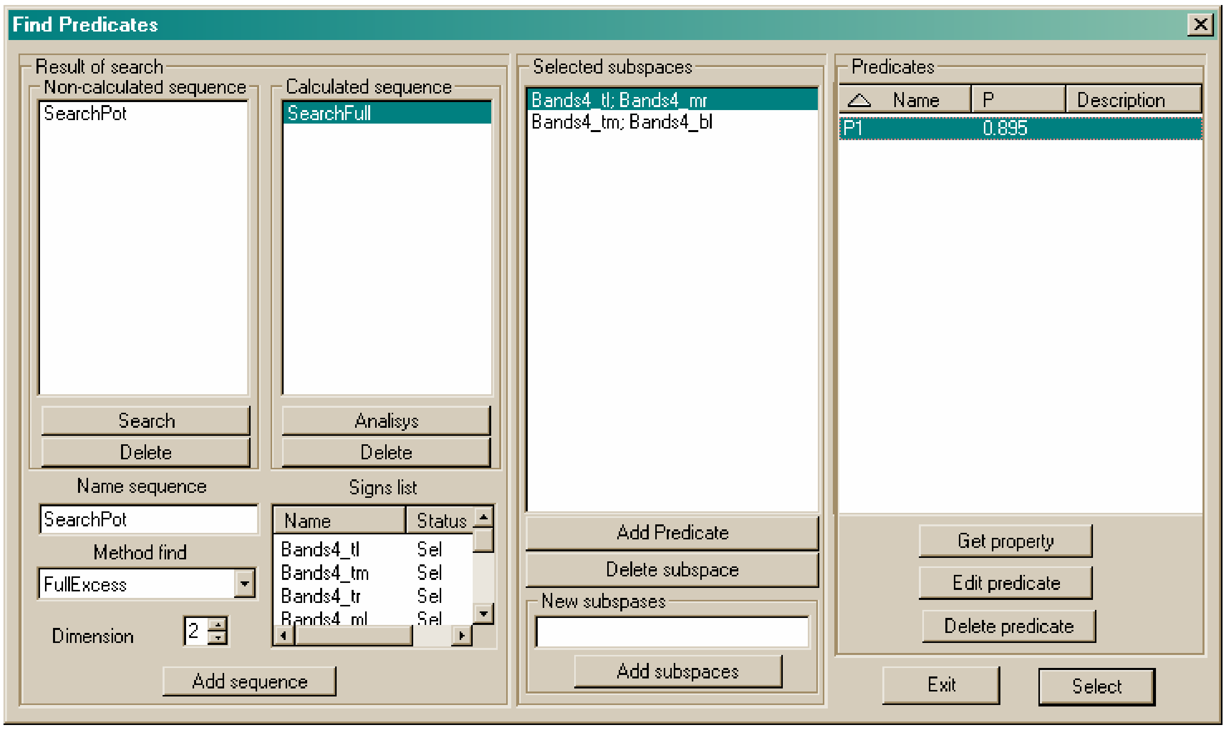


Fig.5.14. Dialog for searching and forming predicates

preliminary results can be saved in order to continue again at the next session of the program operation).

The following actions need to be completed in order to form a particular search task:

- specify the name in the field *Name sequence*;
- specify the dimension of the sought subspace in the field *Dimension*. Only subspaces of dimensions 1 and 2 are suitable for visual analysis. For subspaces of larger dimensions, the formula for separation surface is formed automatically but the results of such “automation” have not yet analyzed or validated because this takes a lot of efforts with regard to a task that is not in the Project focus;
- select the subset of numerical features on which the subspace of the specified dimension will be searched for, in the list *Signs list* (by default, all signs are selected);
- select one of the methods for ordering of the subspaces according to a selected measure of

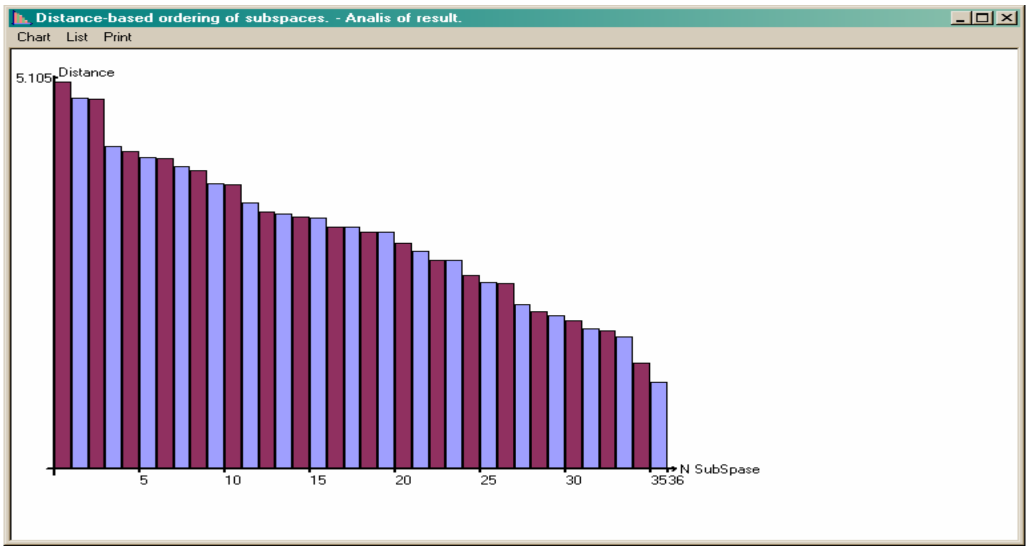


Fig.5.15. Histogram for analysis of the found subspaces

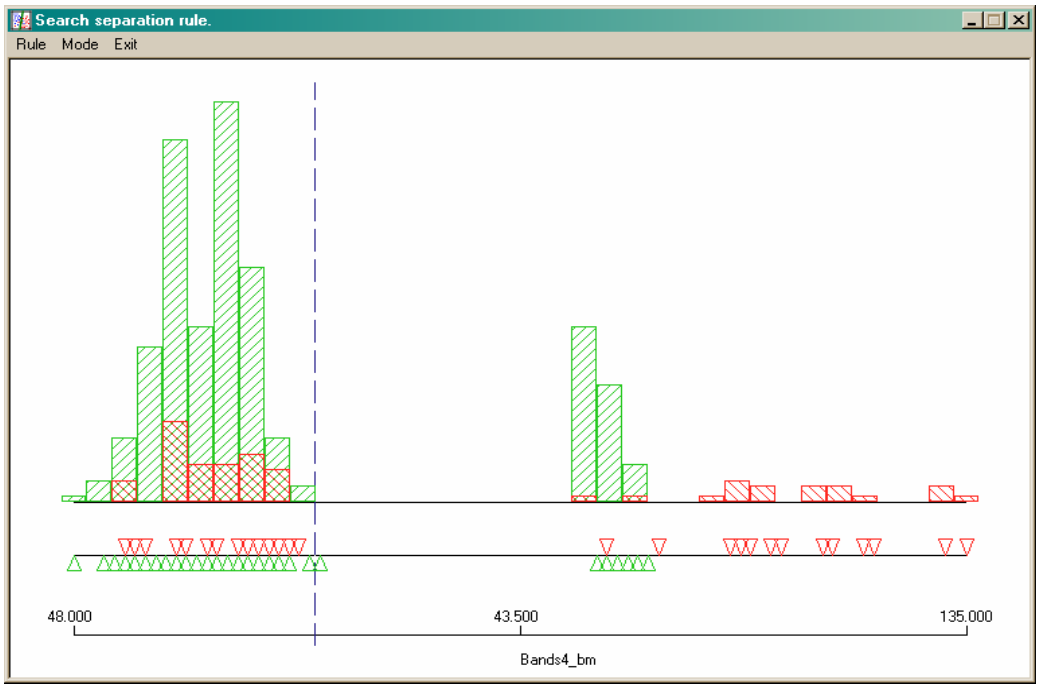


Fig.5.16. Dialog for creating the separation rules in one-dimensional subspaces

informativeness from the ones implemented in the system, in the field *Method find.*
Any search task from the list *Non-calculated sequence* can be executed by clicking on the *Search* button. After the search procedure has been completed, the task will move to the *Calculated sequence* list. The results of search for subspaces for any task from that list can be analyzed by clicking the *Analysis* button. Here, the user will see the interface for analysis of the found subspaces. The user will see a list of subspaces of the specified dimension ordered by the value of interclass distance (the algorithm for calculating the distance is determined by the subspaces search method). The graphical diagram (histogram) of that distribution can be viewed by switching to *Chart* in the menu. An example of the distribution of values of interclass distance in various two-dimensional subspaces is shown as a

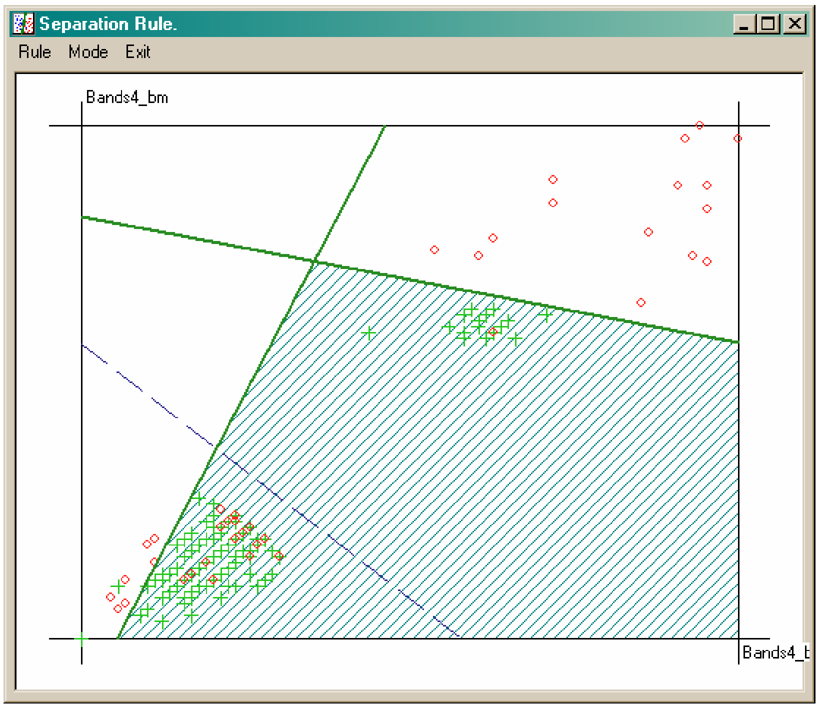


Fig.5.17. Dialog for creating the separation rules in two-dimensional subspaces

diagram in Fig.5.15. By analyzing the search results, user chooses the best subspaces for further analysis. After the analysis window is closed, the selected subspaces are moved to the *Selected subspaces* list. User can also add other subspaces to this list using the *New subspaces* field without activating the search procedure. The subspaces selected for analysis are analyzed individually. Here, the optimal separation line is sought, and the predicate is created.

For analyzing the subspace selected from the list and for forming the predicate, the *Add Predicate* button needs to be clicked on. The interface for creating the separation rules depends on the dimension of the subspace. For one-dimensional subspaces, the interface is shown in Fig.5.16. For two-dimensional subspaces, the separation rules finding interface is shown in Fig.5.17.

After the editing of the separation rule has been completed (or after it has been created automatically for subspaces of dimensions higher than 2), user is asked to enter the name for the newly formed predicate, which is added to the list of designed predicates.

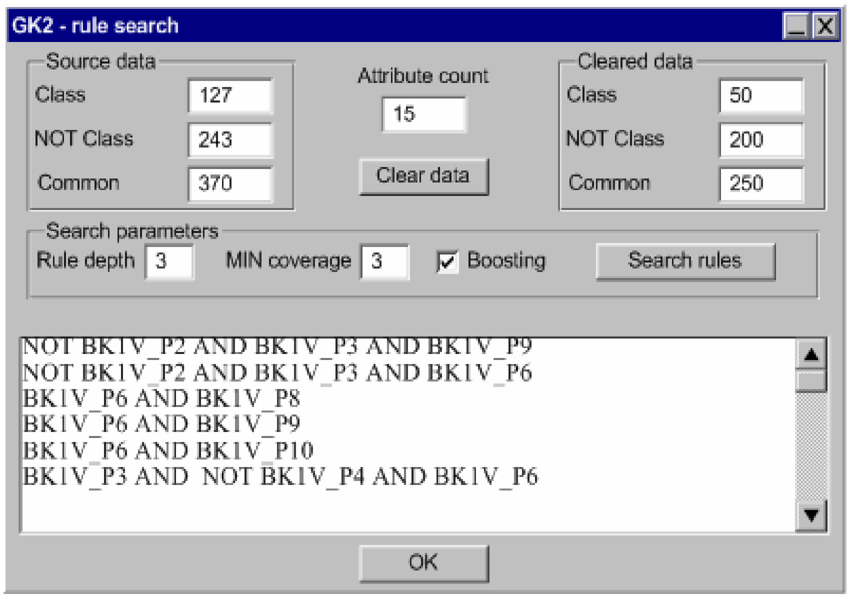
The list of formed predicates shows the obtained predicates and the corresponding probabilities of correct classification. For each predicate, full properties of classification quality can be obtained (based on the *confusion matrix*) by clicking on the *Get property* button. After the analysis of the obtained predicates, user can choose any one of them to be a node of the tree using *Select* button. Here, the list of the tree for which the search was conducted turns into a node, and two leaves are added to it according to the partition of data chosen by the classifier.

After the procedure for creating the predicates has been completed, and the main interface window of *VAM* is activated, the resulting predicates are added to the main list, which is to be used for the final search of the classifier's production rules together with the initial logical attributes of the training sample.

5.3.4. The GK2 Method

The *GK2* module is designed for extraction production rules from the training data sample that consists of logical attributes and predicates found through the *VAM* procedure. The main interface window of the *GK2* module is shown in Fig.5.18.

Group of fields "*Source data"* shows information about parameters of the initial training sample.



Source data		Attribute count	Cleared data	
Class	127	15	Class	50
NOT Class	243		NOT Class	200
Common	370		Common	250

Rule depth 3 MIN coverage 3 ☑ Boosting

```
NOT BK1V_P2 AND BK1V_P3 AND BK1V_P9
NOT BK1V_P2 AND BK1V_P3 AND BK1V_P6
BK1V_P6 AND BK1V_P8
BK1V_P6 AND BK1V_P9
BK1V_P6 AND BK1V_P10
BK1V_P3 AND  NOT BK1V_P4 AND BK1V_P6
```

Fig.5.18. The main interface window of the GK2 module

The rules extraction algorithm utilized in the module only works on consistent data. Data are consistent if it does not contain coinciding data vectors that have different classification within a node. However, the real input data may contain inconsistencies (e.g., due to limited dimension of representation space presented by the vector of features used). Therefore, before the search for rules is initiated, the data needs to be put through the clearing procedure. This procedure is initiated by clicking the *Clear data* button. The properties of data sample ready for training are shown in the group of fields *Cleared data*.

For the rules extraction algorithm's operation, its parameters need to be specified in the group of fields *Search parameters*:

- *Rule depth* - maximum length of the sought rules;
- *MIN coverage* - minimal coverage of the instances of the class by the rule.

The rules finding procedure is initiated by clicking on the *Search rules* button. The found rules are shown in the list *Rules*.

5.3.5. Training Results' Analysis

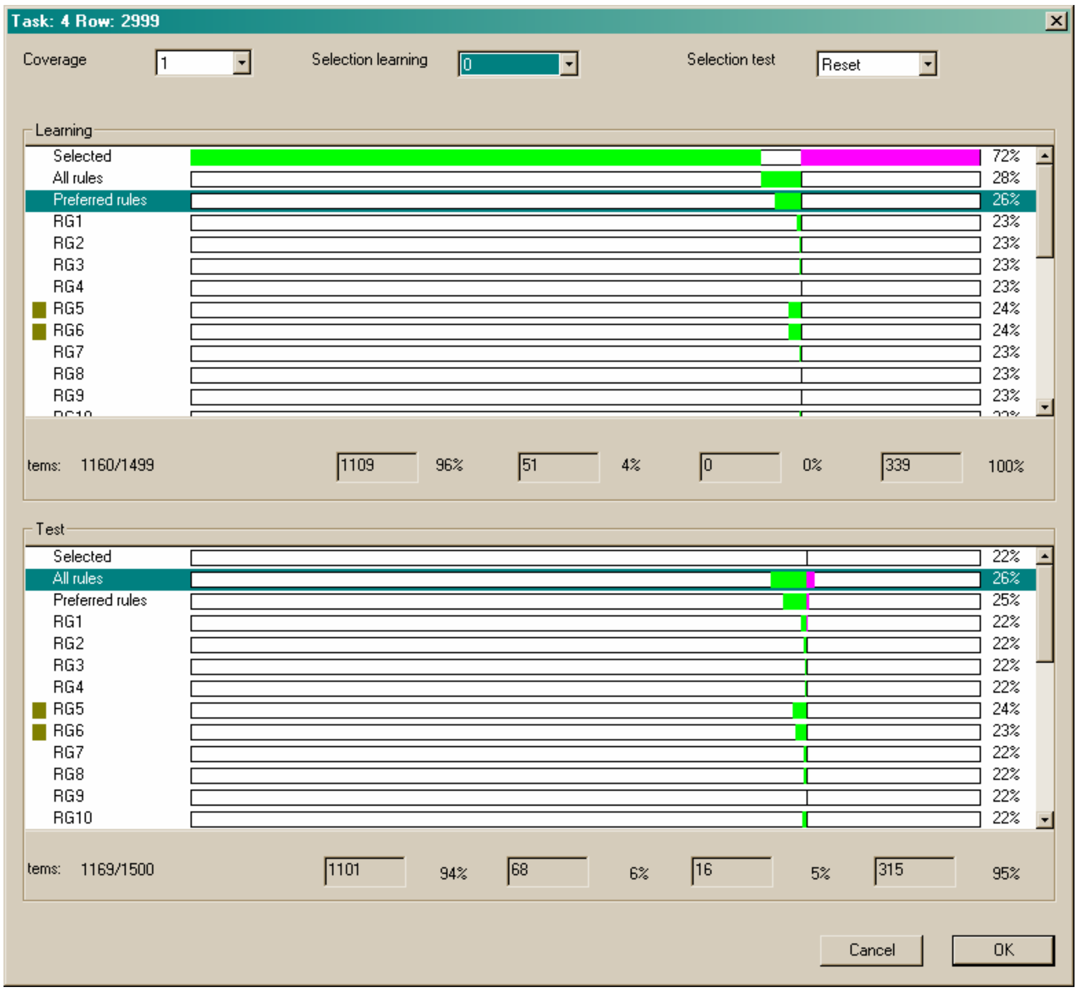
The analysis of the obtained rules is conducted through the dialog whose interface is shown in Fig.5.19. The main tasks solved by user through that interface are:

- Analysis of the quality of the obtained rule, and
- Selection of data for continued training and obtaining new rules.

Rules are analyzed from the standpoint of probabilities of correct classification provided by them, and the coverage of the implementations of both the training and the testing samples. The results of the analysis of obtained rules on the training sample are shown in the top window of the dialog, and the results of analysis on the testing sample – in the lower window of the dialog. Each of these windows shows the complete list of the extracted rules. Here, for each of the rules obtained, the degree of its correct classification for the class and the counter-class separately is shown graphically in a particular table sell. All probabilities from the confusion matrix are also shown for the selected rule. Besides the above results on each rule, the sell *All Rules* shows the results of integrated analysis of the entire set of obtained rules as a whole.

The rule can be included into the classifier's rules base by double-clicking on the appropriate cell.

If it is necessary to continue the training process, the selection of data happens in the following manner. The desired coverage factor of data by the rules is specified in the list Selection Learning. After that, the data (instances) that have the coverage factor in respect to the extracted rules less than or equal to the specified value are automatically selected. Apparently, in the beginning of rules extraction procedure, when no rules have yet been extracted, choosing data with the coverage factor of



Task: 4 Row: 2999

Coverage: 1 | Selection learning: 0 | Selection test: Reset

Learning

Selected	72%
All rules	28%
Preferred rules	26%
RG1	23%
RG2	23%
RG3	23%
RG4	23%
RG5	24%
RG6	24%
RG7	23%
RG8	23%
RG9	23%

tems: 1160/1499 | 1109 | 96% | 51 | 4% | 0 | 0% | 339 | 100%

Test

Selected	22%
All rules	26%
Preferred rules	25%
RG1	22%
RG2	22%
RG3	22%
RG4	22%
RG5	24%
RG6	23%
RG7	22%
RG8	22%
RG9	22%
RG10	22%

tems: 1169/1500 | 1101 | 94% | 68 | 6% | 16 | 5% | 315 | 95%

Cancel | OK

Fig.5.19. The interface for the analysis of the extracted rules

0 will coincide with the entire training sample. The position Selected indicates the results of choosing data for further training on the diagram.

After the interface is closed by clicking on the *OK* button, the obtained data are used for further training, and the selected rules are added to the classifier's rules base.

5.4. Meta-level Intrusion Detection KDD Agent

The main difference between the training of meta–classifiers and the training of base classifiers lies in the fact that their training utilizes data computed by the base classifiers which decisions are combined by the respective meta–classifier. The computation of input data for training and testing of meta–classifiers is the function of the meta–classifier training agent that is *Meta–level KDD agent*. For each meta–classifier training task, this function is represented in terms of state machine for preparing input data and it is initiated in the *Meta–level KDD agent*. A diagram of the state machine states is

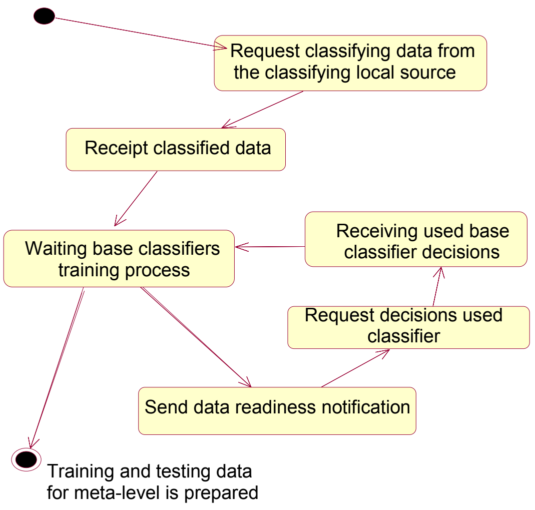

Request classifying data from the classifying local source
Receipt classified data
Waiting base classifiers training process
Receiving used base classifier decisions
Request decisions used classifier
Send data readiness notification
Training and testing data for meta-level is prepared

Fig.5.20. State machine for preparing meta-level input data

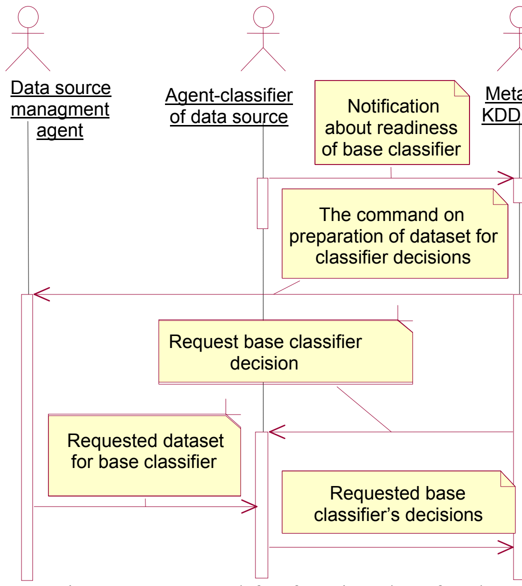

Data source managment agent
Agent-classifier of data source
Notification about readiness of base classifier
Meta-level KDD agent
The command on preparation of dataset for classifier decisions
Request base classifier decision
Requested dataset for base classifier
Requested base classifier's decisions

Fig.5.21. Protocol for forming data for the training of meta–classifier that corresponds to one data source

shown in Fig.5.20.

During the operation of the state machine, meta–classifier training agent requests training data from the *Data source managing agents*. Upon the receipt of that data the state machine goes into waiting state if at least one of the training agents of the respective base classifiers has not completed the training process. After the training of all the necessary base classifiers has been completed, the procedure (protocol) for forming data for the training of meta–classifier that corresponds to one data source shown in Fig.5.21 is initiated. That procedure is executed for each of the base classifiers whose results are being generalized by the meta–classifier in training.

The GK2 component is used for the training of meta–classifier and finding the corresponding rules.

The training results are analyzed through the rules analysis interface used in the analysis of base classifier training.

5.5. DSM-Agents

The main task of the data source management agent is enabling direct access to data and subsequent transformation of the data to the format of the shared application ontology. For that purpose, the data formation state machine is specified in the agent, and its diagram is shown in Fig.5.22.

Each *DSM-agent* enables access to the single data source determined through the name of the database in the ODBC manager. The name of that data source is stored in the agent's database. When

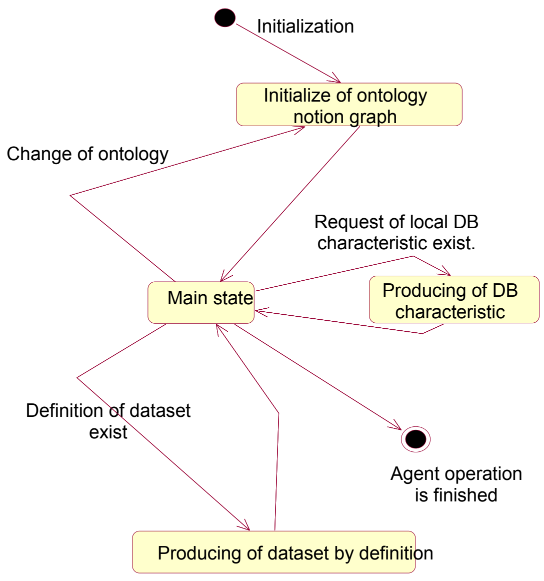

Initialization
Initialize of ontology notion graph
Change of ontology
Request of local DB characteristic exist.
Main state
Producing of DB characteristic
Definition of dataset exist
Agent operation is finished
Producing of dataset by definition

Fig.5.22. Data formation state machine

the agent is started, the state machine is activated that initiates the graph of application domain ontology notions in the memory. Attributes of object domain entities are leaves of that graph. In the nodes of the graph, meta-notions and expressions for their calculations are specified. Connections to the nodes of the lower level indicate the notions of the application domain ontology that are involved in the expression.

The tuning of the ontology of object domain notions of the data access agent is conducted by the database administrator. Here, his/her task is creating the *VIEW* objects in the database with fields that correspond to the attributes of notion (notions) of the object domain, and the notion instance identifier field. The dialog initiated through the command *Open DB Gateway editor* is designed for the tuning of the agent's interface with an external database. The left part of the dialog window shows all entities and attributes of the object domain ontology that are described in the current data source. The right part of the window shows the list of objects of the database and their specifications. Through the appropriate buttons, administrator may establish correspondence between notions and attributes of the application domain ontology and the object of the database.

5.6. Testing of the Designed IDS Prototype and Assessment of Learning Quality

5.6.1. Peculiarities of Training and Testing Data and Respective Procedures

The meta-classification procedure described in Chapter 1 possesses certain peculiarities entailed by the fact that IDS is a real-time system and different base classifiers make their decisions concerning the same connection at different time. This peculiarity entails specificities of both forming meta-learning data and meta-classification (decision making in meta-level) procedures.

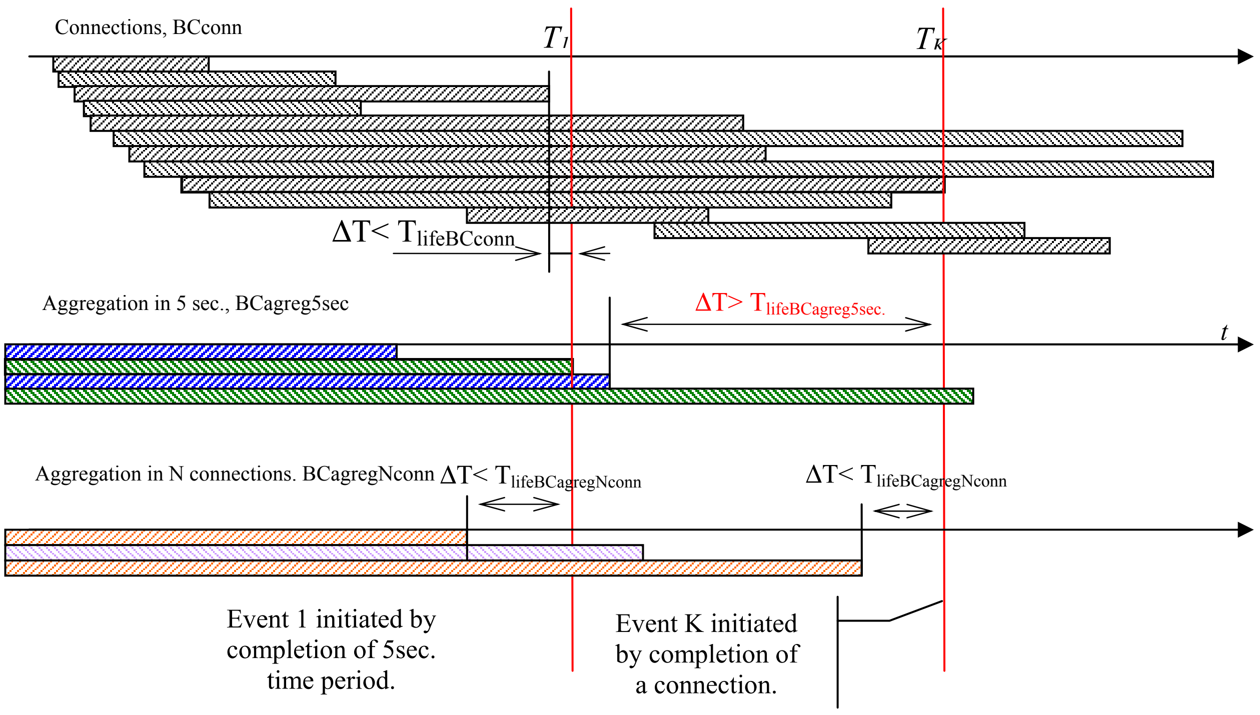
Let us consider the question of how meta-data are computed and what new problems have to be resolved in organization of these computations. While using meta–classification approach, the meta-data is composed of the decisions of the base classifiers which decisions are combined in meta–level by meta–classifier. Since intrusion detection is real-time procedure and output of classifiers are presented as flow of decisions, the *time of occurrence of some events* can be applied as an attribute used for identification of the decisions that has to be composed in a meta-data instance.

An *event* is understood as appearance of new decisions of base classifiers in its output stream of decisions represented in the format <*Decision of base classifier X*, *Time of decision producing*>.

Each base classifier makes its decision at the time when it receives all the data needed for making decisions and it does not produces decision if the required data are incomplete. This means that at a time some base classifiers have already produced decisions but other one not. Therefore, to combine decisions, it is necessary to wait the latter. At the same time, each decision has its own life time and it can be assessed as outdated to the time when other decisions constituting meta–data have made.

Thus, this leads to the following formal model of meta–data:

(1) *K* base classifiers $BC_1 - BC_K$ which output is represented as a stream of decisions at the ordered by random time moments;
(2) Events of the output stream of each particular base classifier are assigned by "life time" $TL_1 - TL_K$ and if it is not used within the respective time interval then the respective event is assigned "Null".


Connections, BCconn
T_1
T_K
$\Delta T < T_{lifeBCconn}$
Aggregation in 5 sec., BCagreg5sec
$\Delta T > T_{lifeBCagreg5sec.}$
t
Aggregation in N connections. BCagregNconn $\Delta T < T_{lifeBCagregNconn}$
$\Delta T < T_{lifeBCagregNconn}$
Event 1 initiated by completion of 5sec. time period.
Event K initiated by completion of a connection.

T	Value BCconn	Value BCagreg5sec	Value BCagregNconn
T_1	*Value exists*	*Value exists*	*Value exists*
T_K	*Value exists*	***NULL***	*Value exists*

Fig.5.23. Explanation of the model of meta-data for training and testing of meta-classifier

Explanation of the above model can be found in Fig. 5.23. This means that it is necessary to use a particular way to compose the decisions in the instance of meta-data. In general case, this question should be carefully analyzed but we will use the simplest of strategy. It is as follows:

If an event occurs in a base classifier then all other ones *submit* to meta-level its *latest decisions* of their decision streams. At that, if $T - T_{event\ i} \le TL_i$ (*i*-the number of base classifier, *T*–the current value of time, *TL*–some "outdating time" constant) then the decision of the respective classifier is missed (it is outdated).

Let us use this strategy of composing meta-data from particular base classifiers' decisions.

In the developed Meta-model of decision fusion, different classifiers are presented. Their peculiarities are briefly as follows:

(1) For classifiers that use information of network level (stream of packet headers, data aggregated per 5 seconds and aggregated data per 100 connections): an event occurs when connection is completed. In turn, completion of such an event can occur in two cases that are (1) the connection is completed normally and Operating system fixes this completion and (2) it is "completed" due to timeout after same waiting period. The second case takes place for *SYNFlood* attack presented in the case study. Since duration of a particular connection corresponding to *SYNFlood* attack can be long enough (as compared with the dynamics of connections stream) and its particular connection looks like normal connection it does not influence on the decisions of the respective base classifiers. This attack can be only detected in higher levels of data abstraction considering attacks carrying out in several connections.

(2) For classifiers that make decisions of the basis of aggregated data per a number of connections: In these classifiers input events occur at the time of beginning of a connection and classifiers' capabilities of various attack detection depends on the "width ΔT of the sliding window" and shift Δ In the case study used in this Project $\Delta T = 100$ and $\Delta = 1$ for the data sample of network-based level and $\Delta T = 30$ and $\Delta = 3$ in application-based level.

(3) For classifiers that make decisions of the basis of aggregated data per time period: their capabilities of various attack detection depend on the width ΔT of time window and shift Δ. In the case study used in this Project ΔT=5 seconds but Δ is variable in terms of time but this time is equal to the time interval between two adjacent connections.

(4) For meta-classifier, an important note concerns to the assumptions used in computing of meta–data used for training and testing. They are as follows:

–If decision of a base classifier is absent (NULL) then such meta–data is excluded from training and testing sample of meta-data. We are forced to accept this assumption because the software capable to mine data with missing values is absent in the library of training and testing data[1];

–In the strategy used in the developed Prototype outdating time TL_i is equated to eternity for all base classifiers.

(4) In each instance of meta–data additional Boolean features are added that indicate an event triggered the meta–classifier to make a decision and they can be of three types, i.e. initiating meta–classifier to make decision. For example, the senses of these features in network-based level are as follows:

– InitConn – decision making of base classifier was triggered by connection completion;

– Init5sec – decision making of base classifier was triggered by completion of time interval of 5 second;

– Init100conn – decision making of base classifier was triggered by completion of interval containing 100 connections.

5.6.2. Description of Training and Testing Results and Evaluation of Classification Quality

Let us comment the results of training and testing carried out by the developed software prototype of IDL MAS. Let us start from such comments for the base and meta–classifiers of the *network level*.

Meta–classifier of this level uses the decisions of three base classifiers that are *BK_ConnAgreg*, *BK_ConnPacket* and *BK_Agreg5sec*. The objective of the classification on the basis of the network-based data sources is to distinct "*Normal"* connections from "*Abnormal*" ones (see Fig.5.24). The resulting probabilities of perfect classification of the best base classifier are 0.73 (over training sample) and 1,0 (for the testing sample). At that, the same probabilities for meta–classifier are 0.81 and 1,0 respectively. Let us comment the cases of *false positives* (missing of attacks) of the meta–classifier. Over the training sample this probability is equal to 0,04 and for the testing sample it is equal to 0,02. The total number of cases in which meta–classifier made such incorrect decisions is equal to 15 (over both training and testing samples). In 11 cases, *FTPCrack* attack was missed but the rest 57 cases of this attach were detected by meta-classifier as suspicious connections that is why these cases are determined correctly as *Abnormal*. The same concerns to 1 case *SYNFlood* attack. 3 cases of the *PipeUpAdmin* attack were completely missed by meta-classifier handling with network level data sources. This means that it is necessary to use other than network level data sources, for example, host-based data source (audit trail of Operating System). An additional note is that this attack practically has to be considered as preparation phase to an attack carried out according to a scenario in which this attack is a step. Network-based level presents no evidences of *PipeUpAdmin* attack.

Let us further comment the training and testing results of the base and meta-level classifiers which make decisions of the basis of the data sources of the application-based level, particularly on the basis of data sources representing performance *FTP* server (see Fig.5.25). Note that these data sources were used to provide the IDS system with the capability to detect the *FTPCrack* attack.

The highest values of the probabilities of perfect classifications provided in this level by a base classifier are equal to 0,86 (over the training sample) and 1,0 (over the testing sample). At that the respective values provided by the meta-classifier of the application-base level are 0,87 and 1,0 respectively.

[1] A technique that is capable to learn from data with missing values is developed and tested but not implemented in the form required to be included in the library.

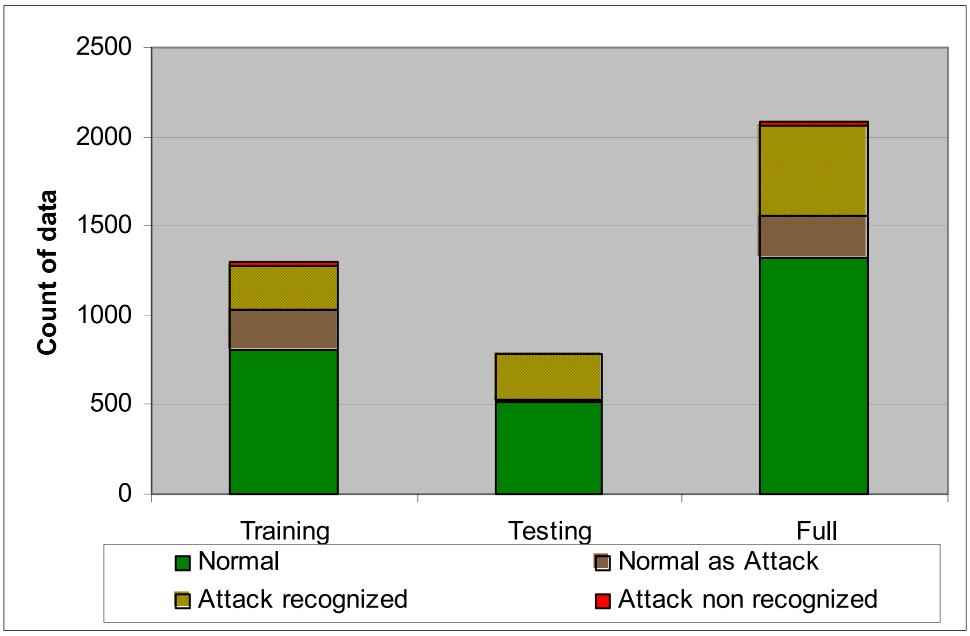


Fig.5.24. Software prototype developed on the basis of network layer data for Abnormal recognition: The probability of perfect classification of the IDS on testing dataset of size of 789 is equal to 0,98

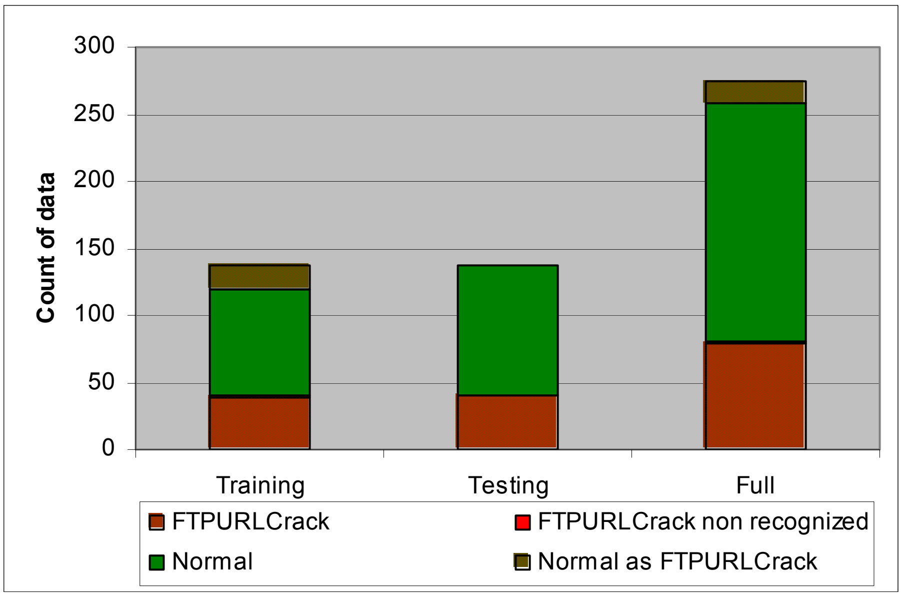


Fig.5.25. Software prototype developed on the basis of FTP server's data for FTPCrack attack recognition: The probability of perfect classification of the IDS on testing dataset of size of 138 is equal to 1,00

Let us comment the cases of false negatives (missing of the attack cases) by meta–classifier. The total probability of false positives is equal to *0,03* over the training sample and 0 over the testing one. In total 1 case was missed. It corresponds to the *first connection* of those composing this attack. The attack is determining by this time after the second connection. Since *FTPCrack* cannot be completed by the first connection then this cannot be completed this attack can be determined in advance thus making it possible to prevent its dangerous consequences.

The total number of false positives (false alarms) is equal to 17 and all of them occurred on training data sample. Thus, the probability of false positives is equal to 0,18 over training data sample and 0 over testing one. All these cases correspond to 8 connections which *immediately following FTPCrack* attack being performing from the same hosts as the hosts from which these attacks were performed.

5.7. Conclusion

The Chapter describes the implemented software prototype implementing the main components of IDLS and presents some simulation results demonstrating the practical use of the developed IDL methodology, technology and supporting software tool.

According to the technology implemented in order to build an IDL MAS (including IDS and IDLS) as applied Information Fusion MAS, the former is firstly specified in the *System Kernel* of the MASDK by making use of Generic agent as a template for specification of agent classes. In parallel problem domain ontology of IDL is also specified. Next step consists in specification of the IDS and IDLS agent classes and the shared component of application ontology. Then agent classes are replicated into agent classes' instances and installed in predefined computers. The resulting IDLS has to be further "filled in" by particular content (data and knowledge interpreting particular ontology notions, providing particular agent procedures with concrete data). After that the IDL MAS operates in the environment independently of MASDK. In the learning mode, training and testing of IDS classification and decision combining agents is fulfilled.

In accordance with the given data sources, the configuration of the software agents' instances of the IDL MAS software prototype is formed in the following manner: (1) For each data source, one logical host and three instances (according to the number of data sources) of software agents *DSM, BC, KDD agent* are specified; (2) To specify the meta–level component of IDL MAS, one or several logical hosts are specified. In each logical host, one instance of software agents *MC* and *KDD local* is specified; (3) Agents of class *KDD Master* that support the management of training and decision making processes are located on the same logical host.

The *base classifier training* scenario by the *KDD–agent of a source* consists of a number of particular subtasks performed in a certain order: Scales conversion to the scales, for which implemented algorithms exist (if the training data contains the attributes of the ordinal or categorical type); Search for rules in favor of class; Tuning of the decision-making mechanism; Testing of classifier; Sending classifier's description to the agent of base classifier.

The *training of meta classifiers* is based on usage of data computed by the base classifiers which decisions are combined by the respective meta classifier. The computation of input data for training and testing of meta classifiers is the function of the *Meta level KDD agent*.

The main task of the *data source management agent* is enabling direct access to data and subsequent transformation of the data to the format of the shared application ontology.

In experiments with the software prototype of IDL MAS components the following *meta-classification and data fusion model* is realized: (1) *K* base classifiers are used which output is represented as a stream of decisions at the ordered by random time moments; (2) If an event occurs in a base classifier then all other ones *submit* to meta-level its *latest decisions* of their decision streams; (3) Events of the output stream of each particular base classifier are assigned by some "life time" and if it is not used within the respective time interval then the respective event is assigned "Null".

The analysis of the testing results of the developed software prototype allow to conclude that the agent-based approach to IDL and developed methodology, technology and software tool constitute a promising starting platform for further research and development of the prospective IDLS. The developed software prototype of IDL MAS showed, for instance, the following results: (1) The probability of perfect classification on testing dataset of size of 789 on the basis of network layer data for Abnormal recognition is equal to 0,98; (2) The probability of perfect classification on testing dataset of size of 138 on the basis of OS and application layers for FTPCrack attack recognition is equal to 1,00.

Project Conclusion

The objectives of the Project, as it was formulated in the Work Plan, were development of the formal model, architecture, and software prototypes of the basic components of the intelligent Multi-agent Learning System intended to provide adaptability of the intrusion detection system (IDS) to the unknown attacks against computer network.

In other words, the Project objective was to find the answer on the question: *Which advantages are provided by use of the multi-agent technology in intrusion detection learning system and to prove the answer via practical development the respective mathematical basis, architecture and technology for intrusion detection learning* (IDL). The preliminary motivation of use of multi-agent technology in the scope of interest is given in the introductory chapter of this Report that is Chapter 1.

We focused our research on the development of such components of security systems that provide the possibility "to learn detection of new attacks and counter-measures in a semi-automatic mode in order to eliminate, as much as possible, the manual and ad-hoc elements from the process of building an intrusion detection system" [Lee-98]. Thus, it should be emphasized that one of the main requirements to intrusion detection rules formed by such learning components is supporting identification of novel attacks and also exhibition of a low false positive rate.

The contemporary studies on data mining for intrusion detection and IDLS prototyping show that existing approaches and techniques cannot completely cover the needs of IDL and one of the most promising approaches to IDSs development is to consider them as a particular case of data and information fusion systems. An important peculiarity of such a view of intrusion detection is that computer network security situational awareness results from composition of decisions produced on the basis of particular data separately providing only partial awareness. But there is lack of researches which practically follow this paradigm. Our research is exactly focused on the development of *IDLS components based on use of data and information fusion principles and built as multi-agent system*.

Formally, IDL task considered in the Project as an application of the general Knowledge Discovery from Databases (KDD) and data and Information Fusion (IF) problem, but it is very specific and differs in many respects with regard to the most "traditional" KDD and IF applications. The main specific properties are as follows:

1. Formidable diversity of attacks (a great deal of existing attack types and diversity of ways of their implementations, increasing number of newly being invented attacks);

2. Multiplicity and diversity of data sources reflecting user's activity (information can be got from numerous heterogeneous sensors monitoring input traffic, audit trails, operational system, servers, applications, directories, databases of user profiles, etc.);

3. Large size and dimensionality of learning data (sensors measure and/or compute numerous characteristics in high frequency real-time mode);

4. Diversity of data sources from several viewpoints (*IP*-packets and their components, symbolic data measured or computed in categorical, Boolean and real-valued scales, temporary ordered sequences and subsequences of events with many attributes, data represented at different generalization level, data derivative due to raw data preprocessing and also high-level computations);

5. Data coherency problem that is understood as the necessity to identify the records of data of different sources associated with the same connection and representing the same "example" of user's activity (e.g. associated with the same attack).

Thus, in the Project, IDS is considered as multisensor knowledge-based IF system. The respective IDL is considered as distributed multi-level data mining and knowledge discovery problem to be implemented on the basis of multi-agent architecture.

The following particular results have been planned to receive within the scope of the Project:

- learning task ontology, allocation of learning tasks over generic learning agents and development of the architecture of their interaction within Multi-agent Learning System;
- mathematical basis and algorithms realizing learning functionalities of the particular agents;

- software prototypes of the components of the Multi-agent Learning System based on theoretical results of the research;
- simulation-based evaluation of the properties, advantages and disadvantages of the developed multi-agent model and architecture of the Multi-agent Learning system aimed to support adaptability and learnability of the Computer Network Security System.

All the tasks provided by the Project and scheduled according to the Work Plan are successfully solved and its results make it possible to positively answer on the question formulated above.

The developed methodology and technology of multi-agent intrusion detection learning based on data and information fusion paradigm can be used in contemporary IDS. It takes into account main specific properties described above.

The *main conclusions based on the results received in the Project* can briefly be formulated as follows:

1. The *main peculiarities of intrusion detection learning technology* result from distributed nature and heterogeneity of audit data (see Chapter 1). The traces of illegitimate activity of users are reflected in multiple distributed and heterogeneous data sources. The data can be represented in different data structures and measured in different measurement scales, be of different accuracy and reliability, they may be incomplete and uncertain, and contain missing values, etc. These properties put specific *problems within IDLS design and implementation* (combination of these problems composes a so-called "data non-congruency problem"): monosemantic understanding of the terminology used by different components of IDLS, entity identification problem, problem of diversity of data measurement scales of training data components, non-coherency of data measurement scales problem, etc.

2. The *peculiarity of our multi-agent technology for IDL* is that it is a specialization of a technology developed for the design and implementation of more general class of information fusion systems (see Chapter 1). The basic components of the methodology concerns such particular components of the data and information fusion systems as (1) Ontology, its roles, structure and its interconnection and communication with distributed data and knowledge bases; (2) Structure of decision making and decision combining specified in terms of Decision Fusion meta–model that is constituted by classification tree (in the top level) and by the set of Decision making trees each of which is mapped to a node of classification tree; (3) Structure of distributed knowledge base which is constituted spatially distributed decision makers (base classifiers in the bottom level and meta–classifiers destined for decision fusion). Each such a classifier is provided with a local knowledge base that is structured according to the Decision fusion meta–model. The top level of the distributed knowledge base is constituted by the application ontology considered as meta–knowledge; (4) A multitude of techniques used for training and testing of classifiers constituting Decision making meta–model; (5) Two different techniques used in decision combining (fusion) procedures; (6) Training and testing methodology; and (7) Methodology of Allocation and Management of Training and Testing Datasets. The above components of the methodology of IDL engineering constitute the conceptual basis, determine the necessary algorithms and also generic architecture of the applied multi-agent IDLS.

3. The developed *technology of multi-agent data and information fusion systems engineering, implementation and deployment* supposes that IDS and IDLS (as applied MASs) are designed by use of two software tools that were developed by authors of this Report. These software tools are Multi-agent System Development Kit (MASDK) and Information Fusion Learning Toolkit. These tools compose the set of components destined for support of IDL MAS design technology and thus together they constitute software tool for IDL MAS technology support. The first toolkit, MASDK, mostly supports engineering, implementation and deployment of the reusable components of IDL MAS that weakly depend on the particular application domain. The second one, Information Fusion Learning Toolkit, is responsible for engineering of the domain and application-dependent components of IDL MAS.

One of two main components of MASDK is so-called "*generic agent*" while the second one is composed of a number of editors destined for specialization of "generic agent" according to particular application in design. Use of such an approach to multi-agent system design leads to very flexible technology in which the target MAS is specified formally in a language (this specification is called MAS "*System kernel*") and afterwards deployed (installed) within a computer network. In case of necessity of MAS modification the designers can do this through modifying specifications of the respective components of the system in the *System kernel* with the subsequent re-generating of the software agents. The main peculiarity of the technology part supported by Information Fusion Learning Toolkit is that the latter actually implements a novel kind of IF technology that can be reasonably called "*agent-mediated technology*". This class of technology assumes that design of IDL MAS is performed by distributed collaborating designers which is activity mediated by a number of agents specifically destined for support of collaboration of designers and dismiss them from a number of routine engineering operations.

4. *The training and testing data for IDL* are historical interpreted audit data containing sequences of users' activity ("cases") that can correspond to "*normal*", "*abnormal*" and "*interpreted abnormal*" data (in the last case it is supposed that the class of attack is determined definitely). In order to construct an efficient IDL MAS, it is necessary to utilize an interconnected complex of audit data received from multiple sources and representing data from different levels of generalization (on the network, OS, application, and additional sources levels). Addressing multiple information sources may significantly increase the validity of decisions related to attack detection and network security.

The taxonomies of these data sources can be formed by different tags (see Chapter 1): (1) *The taxonomy, which classifies data sources due to location of source and software generating data*, includes the network-based sources (depending on network layers and used protocols) and host-based sources (represented by operating system audit trail, system logs, and application-related audit data); (2) *The taxonomy, which classifies data sources due to processing level*, consists from primary sources (network traffic, host command (system calls) traffic, etc.), preprocessed sources (*tcpdump* (for packets), preprocessed OS audit trail, system logs, and audit data of different applications), and generalized sources (generated by statistical processing of preprocessed sources); (3) *The taxonomy, which classifies data sources due to an object, with which the data are associated*, is based on the network-based sources (packets, connections, all network traffic) and host-based sources (traffic within a connection, processes, users, files and directories, disks, system registry, etc.).

Four typical structures of data that can be used in IDL task: Time-based sequential (temporal) data, Sequential (ordered) data, Relational (non-sequential) data, and Transactional data. The typical measurement scales of ID learning data are as follows: Binary (or Boolean), Categorical, Linear ordered, and Real.

The complexity of attack detection learning mechanisms can be significantly reduced through the preliminary *analysis and identification of the most representative and informative attributes of computer network users' activities* that are registered in the audit data. Among such attributes are repeated patterns of events, mistyped commands, indications of exploitation of the known vulnerabilities, illegal parameters, irregularities in the network traffic parameters and contents, substantial discrepancies in the values of attributes that characterize the system subjects' operations profile and unexplained problems (see Chapter 1). Involvement of experts at this stage of learning could substantially cut down the pattern search and dimensions of data needed for learning.

5. *The basis of the IDL problems solutions is many-aspect usage of ontology*. Consistent operation of a large scale distributed system, which makes decisions in a knowledge-based fashion, can be provided in case if agents making up the system are able to "understand" each other. The efficient way to achieve mutual understanding of agents is to use ontology-based approach representing the shared knowledge of distributed entities that form the necessary basis for local knowledge bases consistency, distributed knowledge base integrity and correct interpretation of the messages, which entities exchange with (see Chapter 1). The multi-level ontology of the IDL problem unites a structured multitude of basic notions. This ontology encompasses the notions from several subject domain ontologies, namely, "Data Fusion and Data Fusion Learning problem domain ontology", "Intrusion Detection application ontology" and "Intrusion Detection Learning application ontology" (see Chapter 2). The proposed technology for development and implementation of application ontology supports the development of shared and private components of application ontology that are "coherent" with the intrusion detection problem ontology. The ontology developed serves as a basis for design and

implementation of the upper-level representation of distributed knowledge base of IDL MAS. This level of knowledge provides, on the one hand, integrity of the distributed knowledge base, and on the other hand, “mutual understanding” of the agents interacting via message exchange.

6. The multitude of *methods that covers the needs of IDL task* includes methods for combining decisions produced by base-level classifiers on the basis of different data sources containing fragments of information about status of host operation security, and also data mining and knowledge discovery techniques that are used for training and testing of base-level classifiers (see Chapter 1). Although a lot of methods of data mining and knowledge discovery developed for different types of data structure exist, four basic methods were selected. The selection is based on analysis of data structures that can be perceived or computed within a host with the purpose of analysis of this host security status. The selected methods are: *FP-growth* method of frequent patterns and association rules mining aiming at extraction of useful patterns from transactional (sequential) data; *VAM* (Visual Analytical Mining) aiming at mining rules and other kinds of pattern from numerical data; *GK2* algorithm aiming at mining discrete data, and temporal *data mining algorithm* aiming at mining rules from temporal sequences of binary and/or numerical data.

7. In the developed technology Intrusion Detection Learning and Intrusion Detection procedure itself can be considered either as components of a single system possessing off–line learning capabilities, or they can be considered as the tasks of different systems. In the last case IDL system plays the role of an auxiliary system needed only in design and implementation stages of IDS development (see Chapter 3).

The IDS is viewed as a multisensor multi-level IF system. This system makes decisions on the basis of a multi-level model of processing of input data (network input traffic and/or audit data). The learning technology and respective interaction of both components of IDL MAS is developed in depth, implemented and validated. *Multi-agent architecture of IDL MAS* consists of two kinds of components: (1) Local data source components responsible for operating with particular data sources and (2) Meta-level component responsible for coordination of the performance of component of the first type and also for management of creation of global coherent problem ontology, shared and private components of application ontology, and also for combining decisions of source-based classifiers at meta-level. Local data source components of the system comprise the following parts: *Data source managing agent* responsible for design of its own private and shared parts of application ontology and their co-ordination with the problem ontology; *KDD agent* responsible for training and testing of the classification agents of IDS associated with the local data source, learning meta-classifier(s) and/or referee(s) of the local data source; *Local classification agents* producing decisions on the basis of the local data source; *Server (library) of learning method* that comprises a multitude of KDD methods, metrics for evaluation of the learning quality and other functionalities associated with the solving of knowledge engineering tasks; *Local database* and *user interface* providing interactive mode of its operation. The architecture of the meta-level component comprises the following agents: *KDD Master agent* responsible for design and consistency maintenance of global IDS ontology, realization of a protocol of the local ontology coordination, analysis of local source data structures, support for classification tree design, support for combining decision tree design, setting and passing to the respective data sources the KDD tasks to be solved locally, and management of training and testing data; *Meta-level KDD agent* aiming at solving the tasks of training and testing of agent performing the task of-meta-level; *Agent-classifier of meta-level* that stores meta-level knowledge base created by *Meta-level KDD agent*, receives decisions from local source-based agents of DF system and produces top-level decision; *Server (library) of KDD methods that* stores KDD methods, metrics for evaluation of the learning quality and other functionalities needed for operation of *Meta-level KDD* agent; *user interface* providing interactive mode of its operation. In the multi-agent IDLS the “data non-congruency problem” is solved due to usage of *Data source managing agents* (associated with the particular data sources) and *KDD master agent* (which is a component of meta-level part of multi-agent system). The idea of using in IDLS architecture agents of such kinds is new and seems promising.

8. The *case study for IDL* has been specified (see Chapter 4). To generate training and testing data four types of attack categories were selected: Probing; Remote to local (R2L); Denial of service

(DOS); User to root (U2R). The exemplars of attacks selected for case study are SYN-scan, FTP-crack attack, SYN flood, and PipeUpAdmin. We have chosen three data sources for training and testing data: network-based (traffic level), host-based (operating system level) and application-based (FTP-server level). Each data source was represented by four generic data structures: (1) Time ordered sequence of values of binary vectors of parameters specifying significant events; (2) Statistical attributes of particular connections (performance of a user); (3) Statistical attributes of traffic (users' activity) during the short term time intervals; (4) Statistical attributes of traffic (users' activity) during the long term time intervals.

9. The software prototypes of Intrusion Detection Learning Components were implemented (see the Chapter 5). In accordance with the given data sources, the configuration of the software agents' instances of the IDL MAS software prototype is formed in the following manner: (1) For each data source, one logical host and three instances (according to the number of data sources) of software agents *DSM, BC, KDD agent* are specified; (2) To specify the meta–level component of IDL MAS, one or several logical hosts are specified. In each logical host, one instance of software agents *MC* and *KDD local* is specified; (3) Agents of class *KDD Master* that support the management of training and decision making processes are located on the same logical host. The software code is being written in Visual C++, Java, KQML and XML implementing multi-agent IDLS basic components is currently in progress of debugging.

10. The main results, recommendations and conclusions of the developed architecture and mathematical methods implemented were performed for some particular components of IDLS (see the Chapter 5). The detailed description of the simulation procedure and respective intermediate and final results is given in the Section 5.6 and Appendixes 1 and 2. The analysis of the testing results of the developed software prototype allow to conclude that the agent-based approach to IDL and developed methodology, technology and software tool constitute a promising starting platform for further research and development of the prospective IDLS. The developed software prototype of IDL MAS showed, for instance, the following results: (1) The probability of perfect classification on testing dataset of size of 789 on the basis of network layer data for Abnormal recognition is equal to 0,98; (2) The probability of perfect classification on testing dataset of size of 138 on the basis of OS and application layers for FTPCrack attack recognition is equal to 1,00.

Thus, all the tasks supposed by the Project and indicated in the Work Plan are solved. The main Project results are methodology of intrusion detection learning, IDL ontology, mathematical algorithms realizing IDL, multi-agent architecture of IDL MAS, technology destined for IDL, implementation and deployment of software prototypes of the IDLS components, and simulation-based evaluation of the properties, advantages and disadvantages of IDL MAS.

The research according to this Project proved the advantages of use of multi-agent and approach, architecture and engineering technology as applied to the IDL task.

Future research has to concern new phase of opposition of malefactors and computer network assurance systems. The new bias in this area is that the new danger is associated with distributed (in space and time) and stealthy attacks which are currently of great concerns. This bias requires fundamentally new view of intrusion detection algorithms and means. Actually, this new state of the art in attack organization requires development of scenario-based formal models of distributed and stealth attacks as well as approaches to recognition of such attacks. Practically this means that a new dimension has to be added to the model of intrusion detection and intrusion detection learning tasks. To solve this task, advanced architectures, approaches, formal frameworks, models and particular techniques are needed. They must constitute a new phase of the research in the intrusion detection scope.

Publication of the Project Results

1. V.Gorodetski, O.Karsaev, I.Kotenko, A.Khabalov. Software Development Kit for Multi-agent Systems Design and Implementation. International Workshop of Central and Eastern Europe on Multi-agent Systems (CEEMAS-2001), Krakow, Poland, September 2001.
2. V.Gorodetski, O.Karsaev, I.Kotenko, A.Khabalov. Software Development Kit for Multi-agent Systems Design and Implementation. B.Dunin-Keplicz, E.Navareski (Eds.), From Theory to Practice in Multi-agent Systems. *Lecture Notes in Artificial Intelligence*, Vol. # 2296, pp.121-130, 2002.
3. V.Gorodetski. Multi-agent Data Fusion: Design and Implementation Issues. *5th International Conference on Information Fusion* (*Fusion-2002*), Proceedings of the section "AFOSR Information Fusion Initiative". Annapolis, MD, USA, July 8-10, 2002. Abstract
4. V.Gorodetski, O.Karsayev and V.Samoilov. Multi-agent Data Fusion Systems: Design and Implementation Issues. *Proceedings of the 10th International Conference on Telecommunication Systems - Modeling and Analysis*s, Monterey, CA, October 3-6, vol.2, pp.762-774, 2002.
5. V.Gorodetski, I.Kotenko The Multi-agent Systems for Computer Network Security Assurance: frameworks and case studies. IEEE ICAIS-02. IEEE International Conference "Artificial Intelligence Systems". Proceedings. IEEE Computer Society. 2002. P.297-302.
6. V.Gorodetsky, O.Karsaeyv, and V.Samoilov. Distributed Learning of Information Fusion: A Multi-agent Approach. *In Proceedings of the International Conference Fusion 03*. Cairns, Australia, July 2003.
7. V.Gorodetsky, O.Karsaeyv, and V.Samoilov. Multi-agent Technology for Distributed Data Mining and Classification. *In Proceedings of the IEEE Conference Intelligent Agent Technology* (*IAT03*), Halifax, Canada, October 2003.
8. V.Gorodetsky, O.Karsaeyv, and V.Samoilov. Software Tool for Agent-Based Distributed Data Mining. In *Proceedings of the IEEE Conference Knowledge Intensive Multi-agent Systems* (*KIMAS 03*), Boston, USA, October 2003.
9. V.Gorodetsky, O.Karsaeyv, and V.Samoilov. Multi-agent Data and Information Fusion Architecture, Methodology, Technology and Software Tool. Accepted for publication in the book "*Data Fusion for Situation Monitoring, Incident Detection, Alert and Response Monitoring*". E.Shakhbasyn and P.Vallin (Eds.). To be published in Kluwer Academic Publishers.
10. V.Gorodetsky, I.Kotenko, O.Karsayev. The Multi-agent Technologies for Computer Network Security: Attack Simulation, Intrusion Detection and Intrusion Detection Learning. *The International Journal of Computer Systems Science & Engineering*, 2003, No 4, P.191-200.

Additionally, it was published 6 papers in the conferences proceedings and journals in Russian.

References

[Adamo-00] J.-M. Adamo. Data Mining for Association Rules and Sequential Patterns. Springer Verlag. 2000.

[Adamo-01] J.-M. Adamo. Data Mining for Association Rules and Sequential Patterns : Sequential and Parallel Algorithms. Springer Verlag. 2001

[AgentBuilder-99] *AgentBuilder: An Integrated Toolkit for Constructing Intelligent Software Agents*. Reticular Systems, Inc. Revision 1.3, 1999, http://www.agentbuilder.com.

[Agrawal-95] R. Agrawal. Mining Generalized Association Rules. *Proceedings of the 21st VLDB Conference*, Zurich, 1995.

[Ali *et al*-96] K.M.Ali and M.J.Pazzani. Error reduction through learning multiple descriptions. *Machine Learning*, 24(3), 173-202, 1996.

[Amoroso-99] E.Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Traps, Trace Back, and Response*. Intrusion.Net Books (1999).

[Apap *et al*-02] F. Apap, A. Honig, S. Hershkop, E. Eskin, S. J. Stolfo. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. *Proceedings of the Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002)*. Zurich, Switzerland: October 16-18, 2002.

[Bace-00] R.Bace. *Intrusion Detection*. Macmillan Technical Publishing (2000).

[Barbara *et al*-01a] D. Barbara, N. Wu, S. Jajodia, Detecting Novel Network Intrusions Using Bayes Estimators, Proceedings of the *First SIAM Conference on Data Mining*, Chicago, IL, 2001.

[Barbara *et al*-01b] D. Barbara, J. Couto, S. Jajodia, L. Popyack, and N. Wu. ADAM: Detecting Intrusions by Data Mining. *IEEE Workshop on Information Assurance and Security*, 2001.

[Bass-00] T.Bass. Intrusion Detection System and Multisensor Data Fusion: Creating Cyberspace Situational Awareness. *Communication of the ACM*, vol.43, No. 4, pp.99-105. 2000.

[Bay *et al*-00] S.D.Bay and M.J.Pazzani. Characterizing model error and differences. *Proceedings of 17th International Conferenceon machine learning (ICML-2000)*, Morgan Kaufmann, 2000.

[Bee-gent-00] *Bee-gent Multi-Agent Framework.* Toshiba Corporation Systems and Software Research Laboratories. 2000, http://www2.toshiba.co.jp/beegent/index.htm.

[Bekkerman *et al*-03] R. Bekkerman, R. El-Yaniv, N. Tishby, and Y. Winter. Distributional Word Clusters vs. Words for Text Categorization. *Journal of Machine Learning Research*, V.3, March 2003. P.1183-1208.

[Bengio *et al*-03] Y. Bengio, N. Chapados. Extensions to Metric-Based Model Selection. *Journal of Machine Learning Research*, V.3, March 2003. P.1209-1227.

[Birkhoff-63] G.Birkhoff. Lattice Theory. Providence, Rhode Island, 1963.

[Bloedorn *et al*-01a] E. Bloedorn, A. D. Christiansen, W. Hill, C. Skorupka, L. M. Talbot, J. Tivel. Data Mining for Network Intrusion Detection: How to Get Started. *Conference on Data Mining (SDM 2001)*, Chicago. 2001.

[Bloedorn *et al*-01b] E. Bloedorn, L. Talbot, C. Skorupka, A. Christiansen, W. Hill, and J. Tivel. Data Mining applied to Intrusion Detection: MITRE Experiences. *The 2001 IEEE International Conference on Data Mining.* 2001.

[Booch *et al*-00] G.Booch, J.Rumbaugh, and A.Jacobson. *The unified modeling language user guide*. Adison Wesley Longman, 2000, 429 pp.

[Boumph *et al*-00] F.Boumph, O Direnso, *et al*. *XML Applications*. Wrox Press Ltd. 2000, 688 pp.

[Breiman-96] L. Breiman. Stacked regression. *Machine Learning*, 24(1), 49-64, 1996.

[Brodley *et al*-96] T. Lane and C. E. Brodley. Creating and exploiting diversity. *Proceedings of the AAAI-96 Workshop: Integrating Multiple Learned Models for Improving and Scaling Machine Learning*, pp 8-14. 1996.

[Buntine-90] W.L.Buntine. A theory of learning classification rules. Ph.D thesis, University of Technology, School of Computing Science, Sydney, Australia, 1990.

[Cabrera *et al*-01] J. B. D. Cabrera, L. Lewis, X. Qin, Wenke Lee, Ravi Prasanth, B. Ravichandran, and Raman Mehra. Proactive Detection of Distributed Denial of Service Attacks Using MIB Traffic Variables - A Feasibility Study. *The Seventh IFIP/IEEE International Symposium on Integrated Network Management* (IM 2001), Seattle, WA, May 2001.

[Chan *et al*-98] P.K.Chan and S.J.Stolfo. A comparative evaluation of voting and meta-learning on partitioned data. *Proceedings of Twelfth 4th International Conference on machine Learning,* Tahoe City, CA, 90-98, 1995.

[Chan *et al*-99] P. Chan, W. Fan, A. Prodromidis, and S. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems*, pages 67–74, Nov/Dec 1999.

[Chan *et al*-03] P. Chan, M. Mahoney & M. Arshad. Learning Rules and Clusters for Anomaly Detection in Network Traffic, *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava & A. Lazarevic (editors), Kluwer, 2003.

[Cheswick *et al*-94] W.Cheswick and S.Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacke*r, Addison-Wesley Publishing Company, Reading, MA (1994).

[Clark *et al*-89] P.Clark and T.Niblett. The CN2 Induction Algorithm. *Machine Learning Journal, 3*, pp. 261-283, 1989.

[Clemen-89] R.T.Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5, 559-583, 1989.

[Clifton *et al*-00] C. Clifton, and G. Gengo. Developing Custom Intrusion Detection Filters Using Data Mining. *2000 Military Communications International*, Los Angeles, California, October 22-25. 2000.

[Cohen-95] W.Cohen. Fast effective rule induction. *Machine Learning*: the 12th International Conference, Lake Taho, CA, Morgan Kaufmann (1995).

[Cohen-97] F.Cohen. Information System Attacks: A Preliminary Classification Scheme. *Computers* [Dasgupta-99] D.Dasgupta (Ed).Artificial Immune Systems and Their Applications. Springer Verlag (1999).

[Collis *et al*-99] J.Collis, D.Ndumu. *The Zeus Agent Bilding Toolkit*. ZEUS Technical Manual. Intelligent Systems Research Group, BT Labs. Release 1.0, 1999, http://193.113.209.147/projects/agents/index.htm.

[Cost *et al*-93] S.Cost, S.Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning, 10(1)*, 57-78, 1993.

[Das-00] K. Das. *Attack Development for Intrusion Detection Evaluation*, M.Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 2000.

[Dasgupta *et al*-01] D.Dasgupta and F.Gonzales. An Intelligent Intrusion Detection System for Intrusion Detection. In *Proceedings of the International Workshop "Mathematical Methods, Models and Architectures for Computer Network Security. Lecture Notes in Computer Science,* vol.2052, Springer Verlag, pp.1-14. (2001).

[Denning-87] D.E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, SE-13:222-232, 1987.

[D'haeseleer *et al*-96] P. D'haeseleer, S. Forrest, and P. Helman. An Immunological Approach to Change Detection: Algorithms, Analysis, and Implications. *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy* (1996).

[D'haeseleer *et al*-97] P. D'haeseleer, S. Forrest, and P. Helman. *A distributed approach to anomaly detection* (1997).

[Dietterich-97] T.Dietterich. Machine Learning Research: Four Current Directions. *AI magazine*. 18(4), 97-136, 1997.

[Dietterich-01] T.Dietterich. Ensemble Methods in Machine Learning. In M.Arbib (Ed.) *Handbook of Brain Theory and Neural Networks*, 2nd Edition, MIT Press, 2001.

[Dokas *et al*-02] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava, and P.-N. Tan. Data Mining for Network Intrusion Detection. *Proc. NSF Workshop on Next Generation Data Mining*, Baltimore, MD. 2002.

[Endler-98] D. Endler. Intrusion detection: Applying machine learning to solaris audit data. In *Proceedings of the 1998 Annual Computer Security Applications Conference (ACSAC'98)*, pages 268–279, Los Alamitos, CA, December 1998. IEEE Computer Society, IEEE Computer Society Press. Scottsdale, AZ.

[Eskin-00] E. Eskin. Anomaly detection over noisy data using learned probability distributions. *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, Palo Alto, CA: July, 2000.

[Eskin *et al*-00] E. Eskin, M. Miller, Z.-D. Zhong, G. Yi, W.-A. Lee, and S. Stolfo. Adaptive model generation for intrusion detection. *Proceedings of the ACMCCS Workshop on Intrusion Detection and Prevention*, Athens, Greece, 2000.

[Eskin *et al*-01] E. Eskin, W. Lee, and S. J. Stolfo. Modeling System Calls for Intrusion Detection with Dynamic Window Sizes. *Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX II*), Anaheim, CA, 2001.

[Eskin *et al*-02] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo. A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data. *Data Mining for Security Applications.* Kluwer 2002.

[Fan *et al*-00] W. Fan, W. Lee, S. Stolfo, and M. Miller. A multiple model approach for cost-sensitive intrusion detection. *Proc. 2000 European Conference on Machine Learning* (ECML 2000), LNAI 1810, Barcelona, Spain, May 2000.

[Fan *et al*-01] W. Fan. *Cost-senstive, Scalable and Adaptive Learning Using Ensemble-based Methods*. PhD thesis, Columbia University, Feb 2001.

[Fayyad *et al*-96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process of extracting useful knowledge from volumes of data. *Communications of the ACM*, 39(11), pp. 27-34 (1996).

[Forrest *et al*-94] S. Forrest, A.S. Perelson, L. Allen, R. and Cherukuri. Self-nonself discrimination in a computer. *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA: IEEE Computer Society Press (1994).

[Forrest *et al*-96] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp. 120-128, Los Alamitos, CA, IEEE Computer Society Press (1996).

[Forrest *et al*-97a] S. Forrest, S. Hofmeyr, and A. Somayaji. Computer Immunology. *Communications of the ACM,* Vol. 40, No. 10, pp. 88-96 (1997).

[Forrest *et al*-97b] S. Forrest, A. Somayaji, and D. Ackley. Building diverse computer systems. *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems* (1997).

[Frederick-00] K.Frederick. *Abnormal IP Packets*. www.securityfocus.com (2000).

[Freund *et al*-96] Y.Freund and R.E.Shapire. Experiments with a new boosting algorithm. In L.Saitta(Ed.) *Machine Learning. Proceedings of the 13th International Conference*. Morgan Kaufmann, 1996.

[Gama *et al*-00] J.Gama and P.Brazdil. Cascade generalization. *Machine Learning*, 41(3), 315-342, 2000.

[Ghosh *et al*-98] A.K. Ghosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. *Proceedings of the 1998 Annual Computer Security Applications Conference* (ACSAC'98), December 1998.

[Ghosh *et al*-99a] A. K. Ghosh and A. Schwartzbard. "A study in using neural networks for anomaly and misuse detection". In *Proceedings of 8th USENIX Security.* Washington, D.C, USA, August 23-26, 1999.

[Ghosh *et al*-99b] A.K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*. USENIX Association, April 11-12, 1999.

[Ghosh *et al*-99c] A.K. Ghosh, A. Schwartzbard, and M. Schatz. Using program behavior profiles for intrusion detection. *Proceedings of the SANS Intrusion Detection Workshop*, February 1999.

[Gomez *et al*-03] J. Gomez, D. Dasgupta and F. Gonzalez. Detecting Cyber Attacks with Fuzzy Data Mining Techniques. Proceedings the 2003 SIAM Workshop on Data Mining for Counter Terrorism and Security, San Fracisco, May 1-3, 2003.

[Gomez *et al*-02a] J. Gómez and D. Dasgupta. Evolving Fuzzy Rules for Intrusion Detection. Proceedings of the Third Annual IEEE Information Assurance Workshop 2002, New Jersey, June 2002.

[Gomez *et al*-02b] J. Gómez, D. Dasgupta, O. Nasraoui, and F. González. Complete Expression Trees for Evolving Fuzzy Classifiers Systems with Genetic Algorithms and Application to Network intrusion Detection. Proceedings of NAFIPS-FLINT joint conference, pages 469-474, New Orleans, LA, June 2002.

[Goodman *et al*-97] I.Goodman, R.Mahler, and H.Nguen. *Mathematics of Data Fusion*. Kluwer Academic Publishers, 1997.

[Gorodetski-92] V.Gorodetski. Adaptation Problems in Expert System. *International Journal of Adaptive Control and Signal Processing*, 6, pp.201-210, 1992.

[Gorodetski *et al*-96] V.Gorodetski and O.Karsayev. Algorithm of Rule Extraction from Learning Data. *Proceedings of the 8th International Conference "Expert Systems & Artificial Intelligence" (EXPERSYS-96),* 133-138, 1996.

[Gorodetski *et al*-97] V.Gorodetski, A.Toulupiev. Knowledge Base Consistency Under Interval - Valued Probabilistic Uncertainty. *Transactions of the Russian Academy of Sciences* "*Control Methods and Systems*", 5, Russia, 123-132, 1997. (in Russian)

[Gorodetski *et al*-98] V.Gorodetski, V.Nesterov. Interval Probabilities and Knowledge Engineering. In G.Alefeld and R. Trejo (Eds.) *Proceedings of World Congress on Expert Systems Workshop "Interval Computations and its applications to Reasoning under Uncertainty", Knowledge Representation and Control Theory,* 1998.

[Gorodetski *et al*-00] V.Gorodetski, V.Skormin, L.Popyack, and O.Karsayev. Distributed Learning in a Data Fusion System. *Proceedings of the Conference of the World Computer Congress (WCC-2000) "Intelligent Information Processing" (IIP2000),* 147-154, 2000.

[Gorodetski *et al*-02a] V.Gorodetski, O.Karsayev, I.Kotenko, and A.Khabalov. Software Development Kit for Multi-agent Systems Design and Implementation**.** *Lecture Notes in Artificial Intelligence 2296*, Springer Verlag, 121-130, 2002.

[Gorodetski *et al*-02b] V.Gorodetski and I.Kotenko. Attacks against Computer Network: Formal Grammar-based Framework and Simulation Tool. *Lecture Notes in Computer Science*, V.2516. A.Wespi, G.Vigna, L.Deri (Eds.). *Recent Advances in Intrusion Detection. Fifth International Symposium. RAID 2002.* Zurich, Switzerland. October 2002. Proceedings. Springer Verlag, P.219-238. 2002.

[Gorodetski *et al*-02c] V.Gorodetski, V.Skormin, and L.Popyack. Data Mining Technology for Failure Prognostics of Avionics, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, #2, pp. 388-403, 2002.

[Gorodetski *et al*–02d] V.Gorodetski, O.Karsayev and V.Samoilov. Multi-agent Data Fusion Systems: Design and Implementation Issues. *Proceedings of the 10th International Conference on Telecommunication Systems - Modeling and Analysis*, Monterey, CA, October 3-6, vol.2, pp.762-774, 2002

[Gorodetski *et al*-02e] V.Gorodetski, O.Karsayev. Mining of Data with Missed Values: A Lattice-based Approach. *International Workshop on the Foundation of Data Mining and Discovery in the 2002 IEEE International Conference on Data Mining*, Maebashi TERRSA, Maebashi City, Japan December 9 - 12, 2002, pp.151-156.

[Gorodetski *et al*-02f] V.Gorodetski, V.Skormin, L.Popyack. Data Mining Technology for Failure Prognostics of Avionics, *IEEE Transactions on Aerospace and Electronic Systems*. Volume 38(2), 2002, 388-403.

[Gorodetski-02g] V.Gorodetski. Interim Report #3 on the Project # 1993P. Mathematical Basis of Knowledge Discovery and Autonomous Intelligent Architectures. Task 1. Autonomous Information Collection, Knowledge Discovery Techniques and Software Tool Prototype for Knowledge-Based Data Fusion, November 2002.

[Guyon *et al*-03] I.Guyon, A.Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, V.3, March 2003. P.1157-1182.

[Haines et al-01] J.W. Haines, R.P. Lippmann, D.J. Fried, E. Tran, S. Boswell, and M.A. Zissman. 1999 DARPA Intrusion Detection System Evaluation: Design and Procedures. *MIT Lincoln Laboratory Technical Report TR-1062*, 2001.

[Han *et al*-00] J.Han, J.Pei, B.Mortazavi-Asl, Q.Chen, U.Dayal, and M.Hsu. FreeSpan**:** Frequent pattern-projected sequential pattern mining. *ACM SIGKDD*, 355-359, 2000.

[Han *et al*-01] J.Han and M.Kamber. *Data Mining. Concept and Techniques.* Morgan Kaufman Publishers, 2001.

[Hashem-93] S.Hashem. Optimal linear combination of neural networks. Ph.D. thesis, Purdue University, School of Industrial Engineering, Lafaette, IN. 1993.

[Heller *et al*-03] K. A Heller, K. M. Svore, A. D. Keromytis, and S. J. Stolfo. One Class Support Vector Machines for Detecting Anomalous Window Registry Accesses. *3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security*, Florida, November 19, 2003.

[Hellerstein *et al*-02] Hellerstein, S. Ma, and C.-S. Perng. Discovering actionable patterns in event data. *Systems Journal*, Vol. 41, No. 3, 2002, P.475-493.

[Hershkop *et al*-03] S. Hershkop, R. Ferster, L. H. Bui, K. Wang and S. J. Stolfo. Host-based Anomaly Detection by Wrapping File System Accesses. *CU Tech Report*, April 2003.

[Hofmeyr *et al*-98] S. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security* Vol. 6, pp. 151-180 (1998).

[Hofmeyr *et al*-99] S. Hofmeyr and S. Forrest. Architecture for an Artificial Immune System. *Evolutionary Computation 7(1),* Morgan-Kaufmann, San Francisco, CA, pp. 1289-1296 (1999).

[Honig *et al*-02] A. Honig, A. Howard, E. Eskin, and S. Stolfo. Adaptive Model Generation: An Architecture for the Deployment of Data Minig-based Intrusion Detection Systems. *Data Mining for Security Applications.* Kluwer 2002.

[Howard-97] J.Howard. *An Analysis of Security Incidents on the Internet, 1989 - 199*5, Ph.D. Dissertation, Department of Engineering and Public Policy, Carnegie Mellon University, Pittsburgh, PA (1997).

[Howard *et al*-98] J.Howard, T.Longstaff, *A Common Language for Computer Security Incidents,* SANDIA REPORT, SAND98-8667 (1998).

[http-1] http://193.113.209.147/projects/agents.htm

[http-2] http://www.daml.org/about.html

[http-3] http://www.w3.org/TR/1998/WD-rdf-schema/

[http-4] http://www.ontoprise.de/download/ontoedit_data_sheet.pdf

[http-5] http://protege.stanford.edu/

[http-6] http://www.madkit.org.

[http-7] http://www.kecl.ntt.co.jp/csl/msrg/topics/at/

[http-8] http://www.iks.com/

[http-9] http://www.cognitiveagent.com/

[http-10] http://www.bitpix.com/

[http-11] http://members-http-2.rwc1.sfba.home.net/marcush/ IRS/

[http-12] http://www.objectspace.com/

[http-13] http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm

[http-14] http://samuel.cs.uni-potsdam.de/soft/taxt/research/ade /ade.html

[http-15] http://agent.cs.dartmouth.edu/

[http-16] http://fipa-os.sourceforge.net/

[ideval-99] MIT Lincoln labs. 1999 darpa intrusion detection evaluation, http://www.ll.mit.edu/ist/ideval/ index.html.

[IF] International Society of Information Fusion. The Fusion Problem. http://www.inforfusion.org/ mission.htm.

[InterRep#1] Interim Report #1 on the Project#1994P, Task 2 "Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks", August 2001.

[InterRep#2] Interim Report #2 on the Project#1994P, Task 2 "Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks", May 2002.

[InterRep#3] Interim Report #3 on the Project#1994P, Task 2 "Mathematical Foundations, Architecture and Principles of Implementation of Multi-Agent Learning Components for Attack Detection in Computer Networks", May 2003.

[JADE-99] F.Bellifemine, A.Poggi, and G.Rimassa. JADE – A FIPA-compliant agent framework. *Proceedings of PAAM'99*, London, UK, 1999, http://sharon.cselt.it/projects/jade.

[Javitz *et al*-93] H.S. Javitz, and A. Valdes. The NIDES Statistical Component: Description and Justification, *Technical Report, Computer Science Laboratory, SRI International*, 1993.

[Jordan *al*-94] M.L.Jordan and R.A.Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computations*, 6(2), 181-214, 1994.

[Julisch-01] K. Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. *Proceedings of the 17th ACSAC*, New Orleans, December 2001.

[Julisch-02] K. Julisch. Data Mining for Intrusion Detection: A Critical Review. *Applications of Data Mining in Computer Security*, D. Barbará and S. Jajodia (ed.), Kluwer Academic Publisher, Boston, 2002.

[Julisch *et al*-02] K. Julisch and M. Dacier. Mining Intrusion Detection Alarms for Actionable Knowledge. *The 8th ACM International Conference on Knowledge Discovery and Data Mining*, Edmonton, July 2002.

[Kendall-99] K. Kendall. *A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems*, M.Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1999.

[Kim *et al*-03] H. Kim and P. Chan, Learning Implicit User Interest Hierarchy for Context in Personalization, *Proceedings of International Conference on Intelligent User Interfaces*, p. 101-108, 2003.

[Korba-00] J. Korba. *Windows NT Attacks for the Evaluation of Intrusion Detection Systems*, M.Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, May 2000.

[Krsul-98] I.Krsul, *Software Vulnerability Analysis*, Ph.D. Dissertation, Computer Sciences Department, Purdue University, Lafayette, IN (1998).

[Kubat-96] M.Kubat. Second Tier for Decision Trees. *Machine Learning: Proceedings of the 13th International Conference*, Morgan Kaufman, San Francisco, CA, 1996.

[Kubat *et al*-96] M.Kubat, I.Bratko and R.Michalski (Eds.). A Review of Machine Learning Methods. *Machine Learning and Data Mining. Methods and Applications.* J.Wiley and Sons Ltd, 1996.

[Kumar *et al*-95] S.Kumar and E..Spafford. A software architecture to support misuse intrusion detection. *Proceedings of the 18th National Information Security Conference*, pp. 194-204 (1995).

[Landwehr-94] C.Landwehr, A.Bull, J.McDermott, and W.Choi. A Taxonomy of Computer Security Flaws. *ACM Computing Survey*s, Vol. 26, No. 3, September, pp. 211-254 (1994).

[Lane *et al*-97a] T. Lane and C. E. Brodley. Detecting the Abnormal: Machine Learning in Computer Security. *Technical Report ECE-97-1*, January 1997, Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907.

[Lane *et al*-97b] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*, pp. 43-49. AAAI Press (1997).

[Lane *et al*-97c] T. Lane and C. E. Brodley. An Application of Machine Learning to Anomaly Detection. *20th Annual National Information Systems Security Conference*, Vol. 1, pp 366-380. 1997.

[Lane-98a] T. Lane. Filtering Techniques for Rapid User Classification. *Proceedings of the AAAI-98/ICML-98 Joint Workshop on AI Approaches to Time-series Analysis*, pp 58-63. 1998.

[Lane-98b] T. Lane. *Matching Learning Techniques for the Domain of Anomaly Detection for Computer Security.* Coast TR 98-11. West Lafayette, IN: COAST Laboratory, Purdue University, 1998.

[Lane *et al*-98a] T. Lane and C. E. Brodley. Approaches to Online Learning and Concept Drift for User Identification in Computer Security. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp 259-263. 1998.

[Lane *et al*-98b] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 150–158, 1998.

[Lane-99] T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pp 35-44. 1999.

[Lane *et al*-99] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2:295-331, 1999.

[Lazarevic *et al*-02] A. Lazarevic, P. Dokas, L. Ertoz, V. Kumar, J. Srivastava, and P.-N. Tan. Cyber Threat Analysis - A Key Enabling Technology for the Objective Force (A Case Study in Network Intrusion Detection). *Proceedings 23rd Army Science Conference*, Orlando, FL. 2002.

[Lazarevic *et al*-03a] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. *3rd SIAConference on Data Mining*, San Francisco, CA, 2003.

[Lazarevic *et al*-03b] A. Lazarevic, J. Srivastava, and V. Kumar. Protecting Against Cyber Threats in Network Centric Systems. *SPIE Annual Symposium on AeroSense*, Battlespace Digitization and Network Centric Systems III, Orlando, FL. 2003.

[Lee *et al*-97] W. Lee, S. Stolfo, and P. Chan. Learning patterns from Unix process execution traces for intrusion detection. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pp.50-56. AAAI Press, 1997.

[Lee *et al*-98a] W. Lee, S. J. Stolfo, and K. Mok. Mining audit data to build intrusion detection models. *Proceedings of 4th International Conf. on Knowledge Discovery and Data Mining*, AAAI Press (1998).

[Lee *et al*-98b] W.Lee and S.Stolfo. Data Mining Approaches for Intrusion Detection. *Proceedings 7th USENIX Security Symposium*. 1998. http://www.cs.columbia.edu/~sal/ hpapers/ USENIX/usenix.html (1998).

[Lee *et al*-99a] W.Lee and S.Stolfo, and K.Mok. A Data mining Framework for Building Intrusion Detection Model. *Proceedings of the IEEE Symposium on Security and Privacy,* Oakland, CA, May 1999. IEEE Computer Press, 1999. http://www.cs.columbia.edu/~sal/hpapers/ IEEE99.ps.gz

[Lee *et al*-99b] W. Lee, S. J. Stolfo, and K. Mok. Data mining in work flow environments: Experiences in intrusion detection. *Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99)*, 1999. http://www.cs.columbia.edu/~sal/hpapers/KDD99-id.ps.gz

[Lee *et al*-99c] W. Lee, C. Park, and S. Stolfo. Towards Automatic Intrusion Detection using NFR. *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, April 1999.

[Lee-99] W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, New York, NY. June 1999.

[Lee *et al*-00a] W. Lee and S. Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security,* Volume 3, Number 4 (November 2000).

[Lee *et al*-00b] W. Lee, S. Stolfo, and K. Mok. Algorithms for Mining System Audit Data. *Granular Computing and Data Mining*, T. Y. Lin and N. Cercone (eds), Springer-Verlag, 2000.

[Lee *et al*-00c] W. Lee, S. Stolfo, and K. Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, Kluwer Academic Publishers, 14(6): 533-567, December 2000.

[Lee *et al*-00d] W. Lee, W. Fan, M. Miller, S. Stolfo, and E. Zadok. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. *The First ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.

[Lee *et al*-00e] W. Lee, R. Nimbalkar, K. Yee, S. Patil, P. Desai, T. Tran, and S. J. Stolfo. A data mining and CIDF based approach for detecting novel and distributed intrusions. *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, LNCS 1907. October 2000.

[Lee *et al*-01a] W. Lee and D. Xiang. Information-theoretic measures for Anomaly Detection. In Proceedings of the 2001 IEEE Symposium on Security and Privacy. May, 2001.

[Lee *et al*-01b] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang. Real Time Data Mining-based Intrusion Detection. *Proceedings of Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX II)*. June 2001.

[Liao *et al*-02] Y. Liao and R. Vemuri. Using Text Categorization Techniques for Intrusion Detection. *11th USENIX Security Symposium*, 2002.

[Lindqvist-97] U.Lindqvist and E.Jonsson. How to Systematically Classify Computer Security Intrusions. *Proceedings of the 1997 IEEE Symposium on Security and Privac*y, IEEE Computer Society Press, Los Alamitos, CA, pp. 154-163 (1997).

[Lippmann *et al*-99] R.P. Lippmann, R.K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. W. Webster, M. Zissman, Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation, *Proceedings of the Second International Workshop on Recent Advances in Intrusion Detection (RAID99)*, West Lafayette, IN, 1999.

[Lippmann *et al*-00] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.P. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M. A. Zissman, Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation, *Proceedings DARPA Information Survivability Conference and Exposition (DISCEX) 2000*, Vol. 2, pp. 12-26, IEEE Computer Society Press, Los Alamitos, CA, 2000.

[Lukazky-01] *Intrusion Detection*. BHV, Saint-Petersburg, (2001) (In Russian).

[Luo-99] J. Luo. Integrating Fuzzy Logic With Data Mining Methods for Intrusion Detection, *Master's thesis, Department of Computer Science, Mississippi State University*, 1999.

[Mahoney *et al*-01] M. Mahoney and P. Chan. Detecting Novel Attacks by Identifying Anomalous Network Packet Headers. *Technical Report CS-2001-2*, Florida Tech. 2001.

[Mahoney *et al*-02] M. Mahoney and P. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks, *Proceeding of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 376-385, Edmonton, Canada, July 2002.

[Mahoney-03] M.V. Mahoney. *A Machine Learning Approach to Detecting Attacks by Identifying Anomalies in Network Traffic*. PhD thesis, Florida Institute of Technology, Melbourne, Florida. May, 2003.

[Mahoney *et al*-03a] M. Mahoney and P. Chan. Learning Rules for Anomaly Detection of Hostile Network Traffic. *Proceedings of Third IEEE International Conference on Data Mining*, 2003.

[Mahoney *et al*-03b] M. Mahoney and P. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection, *Proceedings of 6th International Symposium "Recent Advances in Intrusion Detection"*, p. 220-237, 2003.

[Manganaris *et al*-99] S. Manganaris, M. Christensen, D. Serkle, and K. Hermix. A Data Mining Analysis of RTID Alarms. *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection* (*RAID 99*), West Lafayette, IN, September 1999.

[Manganaris *et al*-00] S. Manganaris, M. Christensen, D. Zerkle, and K. Hermiz. A data mining analysis of RTID alarms, *Computer Networks*, 34, 2000, p. 571-577.

[Mannila *et al*-95] H.Mannila, H.Toivonen, and A.Verkamo. Discovering frequent episodes in sequences. *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada (1995).

[McClure *et al*-01] S. McClure, J. Scambray and G. Kurtz. *Hacking Exposed: Network Security Secrets & Solutions*, Third Edition. The McGraw-Hill Companies, Inc. 2001.

[Merz-97] C.J.Merz. Combining classifiers using correspondence analysis. In *Advances in Neural Information Processing*, 1997.

[Merz *al*-97] C.Merz and P.Murphy. UCI repository of machine learning databases. Irvine, CA, University of California, Department of Information and Computer Science, 1997. Available at *http://www.ics.uci.edu/!mlearn/MLR repository.html.*

[Michael-03] C. Michael. Finding the vocabulary of program behavior data for anomaly detection, *Proceedings of DARPA Information Survivability Conference and Exposition III (DISCEX III*), 2003.

[Michalski-83] R.Michalski. A Theory and Methodology of Inductive Learning. J.G.Carbonel, R.S.Michalski, and T.M.Mitchel (Eds.), *Machine Learning,* vol.1, Tigoda, Palo Alto, 83-134, 1983.

[Michalski *et al*-93] The Inferential Theory of Learning: Developing of Foundations of Multi-strategy Learning. *Machine Learning: A Multi-strategy Approach*, *volume 4* (Eds. R.S.Michalski and G.Tecuci), Morgan Kaufman Publishers, 1993.

[Michalski *et al*-97] R.Michalski and A.Kaufman. Data Mining and Knowledge Discovery: A Review of Issues and Multistrategy Approach. *Machine learning and Data Mining: Methods and Applications* (Eds. R.Michalski, I.Bratko and M.Kubat) John Wiley&Sones Ltd., 1997.

[Mukkamala *et al*-00] R. Mukkamala, J. Gagnon, and S. Jajodia. Integrating data mining techniques with intrusion detection methods. *Research Advances in Database and Information Systems Security*, Vijay Atluri and John Hale, editors, Kluwer Publishers, Boston, MA. 33-46. 2000.

[Michalski *et al*-01] Michalski and K.Kaufmann. The AQ19 System for Machine Learning and Pattern discovery: A General Description and User's guide. George Mason University. *Technical Report ML01-2, P01-2,* 2001.

[Mollestad-97] T.Mollestad. A Rough Set Approach to Data Mining: Extracting a Logic of Default Rules from Data. Ph.D Thesis. Tronheim, Norwegian University of Science and Technology, 1997.

[Murthy *et al*-93] S.Murthy, S.Kassif, S.Salzberg, and R.Beigel. OC1: Randomized Induction of oblique decision trees. *Proceedings of AAAI-93*, AAAI Press, 1993.

[Niyogi *et al*-00] P.Niyogi, J-B.Pierrot, O.Siohan. Multiple Classifiers by constrained minimization. *Proceedings of International Conference on Acoustics, Speech, and Signal Processing,* Istanbul, Turkey, 2000.

[Northcutt-99] S.Northcutt. *Network Intrusion Detection. An Analyst's Handbook*. New Riders Publishing (1999).

[Ortega *et al*-01] J.Ortega, M.Coppel, and S.Argamon. Arbitraining Among Competing Classifiers Using Learned Referees. *Knowledge and Information Systems*, 4, 2001, 470-490.

[Pawlak-82] Z.Pawlak. Rough Sets. *International Journal of Computer and Information Sciences*, 11, 341-356, 1982.

[Perrone *et al*-93] M.P.Perrone and L.N.Cooper. When networks disagree: Ensemble methods for hybrid neural networks/ In R.J.Mammone (Ed.), *Neural Networks for Speech and Image Processing*, Chapman and Hall, 1993.

[Pfanzagl-71] J.Pfanzagl. Theory of Measurement. Phyzica-Verlag, Wuerzburg-Wien, 1971.

[Porras *et al*-98] P.Porras and A.Valdes. Live traffic analysis of tcp/ip gateways. *Proceedings of the Internet Society Symposium on Network and Distributed System Security* (1998).

[Portnoy-00] L. Portnoy. Intrusion detection with unlabeled data using clustering. *Undergraduate Thesis*, Columbia University, Department of Computer Science, 2000.

[Portnoy *et al*-01] L. Portnoy, E. Eskin and S. J. Stolfo. Intrusion detection with unlabeled data using clustering. *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Philadelphia, PA: November 5-8, 2001.

[Posland *et al*-00] S.J.Poslad, S.J.Buckle, and R.Hadingham: The FIPA-OS agent platform: Open Source for Open Standards. *Proceedings of PAAM 2000*, Manchester, UK, 2000, http://fipa-os.sourceforge.net/.

[Proctor-01] P.Proctor. *Practical Intrusion Detection Handbook*. Prentice Hall PTR (2001).

[Prodromidis *et al*-99a] A. Prodromidis, S. Stolfo, and P. Chan. Effective and Efficient Pruning of Meta-Classifiers in a Distributed Data Mining System. *Journal on Data Mining and Knowledge Discovery*, 1999. http://www.cs.columbia.edu/~sal/hpapers/DMKDJ.ps.gz

[Prodromidis *et al*-99b] A. Prodromidis, P. Chan, and S. Stolfo. Meta-Learning in Distributed Data Mining Systems: Issues and Approaches, *Advances in Distributed Data Mining*, AAAI Press, Kargupta and Chan (eds.). 1999. http://www.cs.columbia.edu/~sal/hpapers/ DDMBOOK.ps.gz

[Prodromidis-99] A.L. Prodromidis. *Management of Intelligent Learning Agents in Distributed Data Mining Systems. PhD Theses.* COLUMBIA UNIVERSITY. 1999.

[Quinlan-86] J.Quinlan. Induction of Decision Trees. *Machine Learning*, 1, vol.1, 1986.

[Quinlan-92] J.Quinlan. C4.5: Program for Machine Learning. Morgan Kaufmann, 1992.

[Quinlan-93] R.Quinlan. C4.5 Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[Quinlan-96] J.Quinlan. Improved Use of Continuous Attributes in C.4.5. *Journal of Artificial Intelligence Research*, 4, 77-90, 1996.

[Rao-68] C.R.Rao Linear Statistical Inference and its Applications. John Wiley & Sons. Inc. 1968.

[Raseva *et al*-63] H.Raseva and R.Sikorski. The Mathematics of Meta-mathematics. Warshaw, Monograpie Mathematyczne, 1963.

[Report-99] Applied Methods and Models of Knowledge Engineering in Information Based Health Assessment Systems. Final Report on Contract No.F61775-98-WE116, 1999.

[RFC0793-81] RFC 0793. J. Postel. *Transmission Control Protocol* (1981).

[RFC1918-96] RFC 1918. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot & E. Lear. *Address Allocation for Private Internets* (1996).

[RFC0931-85] RFC 0931. M. St. Johns. *Authentication server* (1985).

[Robertson *et al*-03] S. Robertson, E. Siegel, M. Miller and S. Stolfo. Surveillance detection in high bandwidth environments, *Proceedings of DARPA Information Survivability Conference and Exposition III (DISCEX III*), 2003.

[Rumelhart *et al*-86] D.E.Rumelhart, G.E.Hinton, and R.J.Williams. Learning internal representation by error propagation. In D.E.Rumelhart, J.L.McClelland (Eds.) *Parallel Distributed Processing: Exploration of the microstructure of cognitions, Volume 1: Foundations.* MIT Press, 1986.

[Samoilov-02] V. Samoilov. Data Fusion Systems: Principles and Architecture for Data Processing in Decision Making System Learning. *Transactions of SPIIRAS*, # 1, 2002. (In Russian)

[sa2003] NSFOCUS Security Advisory (SA2003-01). http://packetstormsecurity.nl/0303-exploits/sa2003-01.txt

[Scambray et al-01] J. Scambray and S. McClure. *Hacking Exposed Windows 2000*. The McGraw-Hill Companies, Inc. 2001.

[Schultz *et al*-01a] M. G. Schultz, E. Eskin, and S. J. Stolfo. MEF: Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables. *USENIX Annual Technical Conference - FREENIX Track*, Boston, MA, June 2001.

[Schultz *et al*-01b] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.

[Seewald *et al*-01] A.K.Seewald, J.Fuernkranz. An evaluation of grading classifiers. *Proceedings of 4th International Conference "Intelligent data Analysis"*, LNCS 2189, Springer Verlag, 115-124, 2001.

[Semantic Web] *http://www.w3.org/DesignIssu es/Semantic.html*

[Sequeira *et al*-02] K. Sequeira, and M. Zaki. ADMIT: Anomaly-base Data Mining for Intrusions. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, July 2002.

[Skormin *et al*-97] V.Skormin and L.Popyack. Reliability of Avionics and "History of Abuse". A Prognostic Technique. *Proceedings of ICI&C '97,* St. Petersburg, Russia. pp. Lxxvi-lxxxii, 1997.

[Skormin *et al*-01] V.Skormin, J.Delgado-Frias, D.McGee, J.Giordano, L.Popyack, V.Gorodetski, A.Tarakanov. BASIS: A Biological Approach to System Information Security. In *Proceedings of the International Workshop "Mathematical Methods, Models and Architectures for Computer Network Security. Lecture Notes in Computer Science,* vol.2052, Springer Verlag, pp.127-142 (2001).

[Skowron-00] A.Skowron. Rough Sets in KDD. *Proceedings of 16th World Computer Congress, vol. "Intelligent Information Processing"*, Beijing, 1-17, 2000.

[Sloman-98] A.Sloman: What's an AI Toolkit For? In *Proceedings of the AAAI-98 Workshop on Software Tools for Developing Agents*. Madison, Wisconsin, 1998.

[Somayaji *et al*-98] A. Somayaji, S. Hofmeyr, and S. Forrest. Principles of a Computer Immune System. *1997 New Security Paradigms Workshop* pp. 75-82, 1998.

[Srikant *et al*-95] R. Srikant and R. Agrawal. Mining generalized association rules. *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland (1995).

[Stolfo *et al*-97a] S. Stolfo, D. Fan, W. Lee, A. Prodromidis, and P. Chan. Credit Card Fraud Detection Using Meta-Learning: Issues and Initial Results. AAAI Workshop: AI Approaches to Fraud Detection and Risk Management, July 1997.

[Stolfo *et al*-97b] S.J. Stolfo, D. Fan, W. Lee, A. Prodromidis, and P. Chan. JAM: Java Agents for Meta-learning over Distributed Databases. *Proc. KDD-97 (runner up best paper, applications) and AAAI97 Work. on AI Methods in Fraud and Risk Management)*, 1997.

[Stolfo *et al*-97c] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. JAM: Java agents for meta-learning over distributed databases. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (KDD '97), pages 74–81, Newport Beach, CA, August 1997. AAAI Press.

[Stolfo *et al*-00] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX '00)*, Hilton Head, SC, January 2000.

[Stolfo-03] S.J. Stolfo. Behavior-based Computer Security. Lecture Notes in Computer Science, Springer-Verlag, V.2776. Theory and Practice of Computer Network Security. Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security, St. Petersburg, Russia, September 21–23, 2003. P.57-81.

[Stolfo *et al*-03a] S. J. Stolfo, Wei-Jen Li, S. Hershkop, K. Wang, C.-W. Hu, O. Nimeskern. Detecting Viral Propagations Using Email Behavior Profiles. *CU Tech Report*, 2003.

[Stolfo *et al*-03b] S. J. Stolfo, C.-W. Hu, W.-J. Li, S. Hershkop, K. Wang, and O. Nimeskern. Combining Behavior Models to Secure Email Systems. *CU Tech Report*, April 2003.

[Stolfo *et al*-03c] S. J. Stolfo, S. Hershkop, K. Wang, O. Nimeskern, and C.-W. Hu. Behavior Profiling of Email. *1st NSF/NIJ Symposium on Intelligence & Security Informatics (ISI 2003).* June 2-3, 2003, Tucson, Arizona, USA.

[Stolfo *et al*-03d] S. J. Stolfo, S. Hershkop, K. Wang, O. Nimerkern and C.-W. Hu. A Behavior-based Approach to Securing Email Systems. *Mathematical Methods, Models and Architectures for Computer Networks Security*, Proceedings, Springer Verlag, Sept. 2003.

[Suppes *et al*-63] P.Suppes, J.Zinnes. Basic Measurement Theory. In *"Handbook Math. Psychol."* (eds.R.Luce, R.Bush, E.Galanter). New York, vol. 1, pp.1-76, 1963.

[Tcptrace] Tcptrace software tool, www.tcptrace.org

[Teng *et al*-90] H. S. Teng, K. Chen, and S. C. Lu. Adaptive real-time anomaly detection using inductively generated sequential patterns. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 278–284, Oakland CA, May 1990.

[Ting-96] K.Ting. The characterization of predictive accuracy and decision combination. In *Procedings of 13th International Conference on Machine Learning*, Morgan Kaufman, 1996, 498-506.

[Ting *et al*-99] K.M.Ting and I.H.Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research,* 10, 271-289, 1999.

[Todorovski *et al*-00] L.Todorovski and S.Dzeroski. Combining classifiers with meta–decision trees. D.A.Zighen, J.Komarowski and J.Zitkov (Eds.) In *Proceedings of 4th European Conference on Principles of Data Mining and Knowledge Discovery* (*PKDD-00),* France, Springer Verlag, 2000, 54-64.

[Todorovski *et al*-01] L.Todorovski, S.Dzeroski. Combining multiple models classifiers with meta–decision trees. *Machine Learning Journal*. 2001.

[Vilalta *et al*-01] R.Vilalta and Y.Drissi. A perspective view and survey of meta-learning. Submitted to the *Journal of Artificial Intelligence Review* (March 2001). Available at *http://www.research.ibm.com/people/v/vilalta/papers/jaireview01.ps.*

[Wang *et al*-03] K. Wang, S. J. Stolfo. One Class Training for Masquerade Detection. *CU Tech Report*, April 2003.

[Warrender *et al*-99] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. *Proceedings of 1999 IEEE Symposium on Security and Privacy, pages 133-145.* IEEE Computer Society, 1999.

[Weiss *et al*-99] Multiagent Systems. A modern Approach to Distributed Artificial Intelligence. Ed. G.Weiss. MIT Press, 1999, 376 pp.

[Wolpert-92] D.Wolpert. Stacked generalization. *Neural Network*, 5(2), 241-260, 1992.

[Wooldridge-01] M.Wooldridge. An Introduction to Multi-agent Systems. John Wilie&Sons, LTD, 2001,348 pp.

[Ye-00] N. Ye. A Markov Chain Model of Temporal Behavior for Anomaly Detection, *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, 2000.

[Ye *et al*-01] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu. Probabilistic techniques for intrusion detection based on computer audit data. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 31, No. 4, 2001, pp. 266-274.

[Ye *et al*-02a] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on Computers*, Vol. 51. No. 7, 2002, pp. 810-820.

[Ye *et al*-02b] N. Ye, C. Borror, and Y. Zhang. EWMA techniques for computer intrusion detection through anomalous changes in event intensity. *Quality and Reliability Engineering International*, Vol. 18, No. 6, 2002, pp. 443-451.

[Ye *et al*-03] N. Ye, and Q. Chen. Computer intrusion detection through EWMA for auto-correlated and uncorrelated data. *IEEE Transactions on Reliability*, Vol. 52, No. 1, 2003, pp. 73-82.

[Zhang *et al*-00] Y. Zhang and W. Lee. Intrusion Detection in Wireless Ad-Hoc Networks. Proceedings of the Sixth International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, MA, August 2000.

[Zytkov *et al*-00] J.Zytkov, W.Klosgen. *Machine Discovery Terminology*. http://orgwis.gmd.de/ projects/explora/terms.html (2000).

Appendixes. Logs of Operation of the Developed Software Prototype of Multi-agent Learning System: Training and Testing for the Application Corresponding to the Case Study

The purpose of this Appendix is to demonstrate the operation of the developed multi-agent distributed data mining and decision making system prototype (including IDLS and IDS components) and the respective technology as applied to the training and testing dataset from intrusion detection learning area that was developed by authors of this Report[1]. The dataset itself was described in Chapter 5. Let us note that this data corresponds to three different data sources of network level, host-based level and application level (but in experiments fulfilled two last levels were combined). Additionally, each above mentioned data source includes temporal and statistical data calculated on different (short term and long term) basis. Let us also note that the necessity to develop new training and testing dataset was entailed by the fact that didn't succeed in finding anywhere a dataset of the more or less complicate structure that contain data sources of different levels and several data types in each of them.

Appendix A. Training and Testing on the Basis of Network-based Datasets

1. Classes and Data Sets

1.1. Structure of classes

- *Normal*: Connection Status ='*Normal*';
- *Abnormal:* Connection Status ='NOT *Normal*', which includes *Abnormal* user activities and attacks.

Classification tree includes *Root node* with 2 branches leading to the nodes marked as *Normal* and *Abnormal.*

1.2. Analysis of datasets in the source NetLevel – Connections

Number of cases – 432.

ID_ENTITY	432
Normal	114
Abnormal	318

Assignment of data for training and testing of base classifiers:
- Connection.ID_ENTITY<= 'Thu Jul 17 21:27:54,960745 2003' AND *Normal*
- Connection.ID_ENTITY<='Thu Jul 17 22:01:34,172798 2003' AND *Abnormal*

Number of cases of the classes:
Normal – 57;
Abnormal – 157.

Conditions for selection of training data:
Percentage 100%: (Connection.ID_ENTITY <= 'Thu Jul 17 21:24:39,126390') AND *Normal*
Percentage 50%: (Connection.ID_ENTITY <= 'Thu Jul 17 22:01:34,172798 2003') AND *Abnormal*

Number of cases of the classes:
Normal – 29;
Abnormal – 78.

Conditions for selection of training data:
Percentage 100%: ((Connection.ID_ENTITY > 'Thu Jul 17 21:24:39,126390') AND (Connection.ID_ENTITY<= 'Thu Jul 17 21:27:54,960745 2003')) AND *Normal*

[1] This dataset can be made available to the Partner (EOARD) on request.

Percentage 50%: (Connection.ID_ENTITY <= 'Thu Jul 17 22:01:34,172798 2003') AND
Abnormal
Number of cases of the classes:
Normal – 28;
Abnormal – 78.

2. Training of the base classifier *BKConnectionNormal*

While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative:
- Connection_SYN_src_dst

2.1. Extraction rules of class *Normal*

Search for predicates by use VAM algorithm:

Result: 6 rules:

```
BKConectionNormal_N1= (+1603.4361861057725000*Connection.Packets-
20773.3892031407300000 > 0) AND (+0.9999874021551684*
Connection.Duration+0.0050195150121859*Connection.PSH_dst_src-
0.0668631266297002 > 0)

BKConectionNormal_N2=(+0.3898192994799297*Connection.Packets-
0.9208913691380632*Connection.PSH_src_dst-3.0964780702130232 > 0) AND
(+0.9997786108798298*Connection.Duration-0.0210411317946090*
Connection.PSH_dst_src-0.0408048178738170 > 0)

BKConectionNormal_N3=(+0.9892747788830054*Connection.Duration+0.14606646386
48474*Connection.Packets-1.8915060806311901 > 0) AND
(+0.9999469760542993*Connection.Duration+0.0102978191799450*Connection.PSH_
dst_src-0.0678848187709065 > 0)AND (+0.9974120304999046*
Connection.Duration-0.0718974367697309*Connection.PSH_dst_src+
0.6218769870346778 > 0)

BKConectionNormal_N4=(-0.2660029503134025*Connection.Duration+
0.9639722145500697*Connection.FIN_src_dst-0.1513078382144927 > 0) AND
( (+0.9934378934982687*Connection.Duration-0.1143728628728094*
Connection.PSH_src_dst+0.9060659801972677 > 0)AND (-
0.6627664212686979*Connection.Duration-0.7488261953475471*
Connection.PSH_src_dst+5.5152290587675612 > 0)AND NOT(-
0.9176616247245791*Connection.Duration+0.3973627341709909*Connection.PSH_sr
c_dst-0.3403521618557532 > 0) ) OR ((+0.9934378934982687*
Connection.Duration-0.1143728628728094*Connection.PSH_src_dst+
0.9060659801972677 > 0)AND NOT(-0.6627664212686979* Connection.Duration-
0.7488261953475471*Connection.PSH_src_dst+ 5.5152290587675612 > 0)AND (-
0.9176616247245791*Connection.Duration+
0.3973627341709909*Connection.PSH_src_dst-0.3403521618557532 > 0))

BKConectionNormal_N5=(+1666.6099790907281000*Connection.Packets+262.1039945
892520100*Connection.PSH_dst_src+231.3465640879086600*Connection.PSH_src_ds
t-22684.4403668937520000 > 0)

BKConectionNormal_N6=(-0.2057401826907004*Connection.Duration+
0.3950037687846065*Connection.FIN_dst_src+0.3950037687846065*Connection.FIN
_src_dst+0.3841303265549015*Connection.SYN_dst_src+0.0372738357257134 > 0)
AND (+0.0074761790042960*Connection.Duration+
0.2615179611123543*Connection.FIN_dst_src+0.2615179611123543*Connection.FIN
_src_dst-0.2868432626062151*Connection.Status+ 0.2361926596184932
*Connection.SYN_dst_src+0.0834705042691872 > 0)
```

Search for predicates by use GK2 algorithm.

While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative

GK2 algorithm attributes:

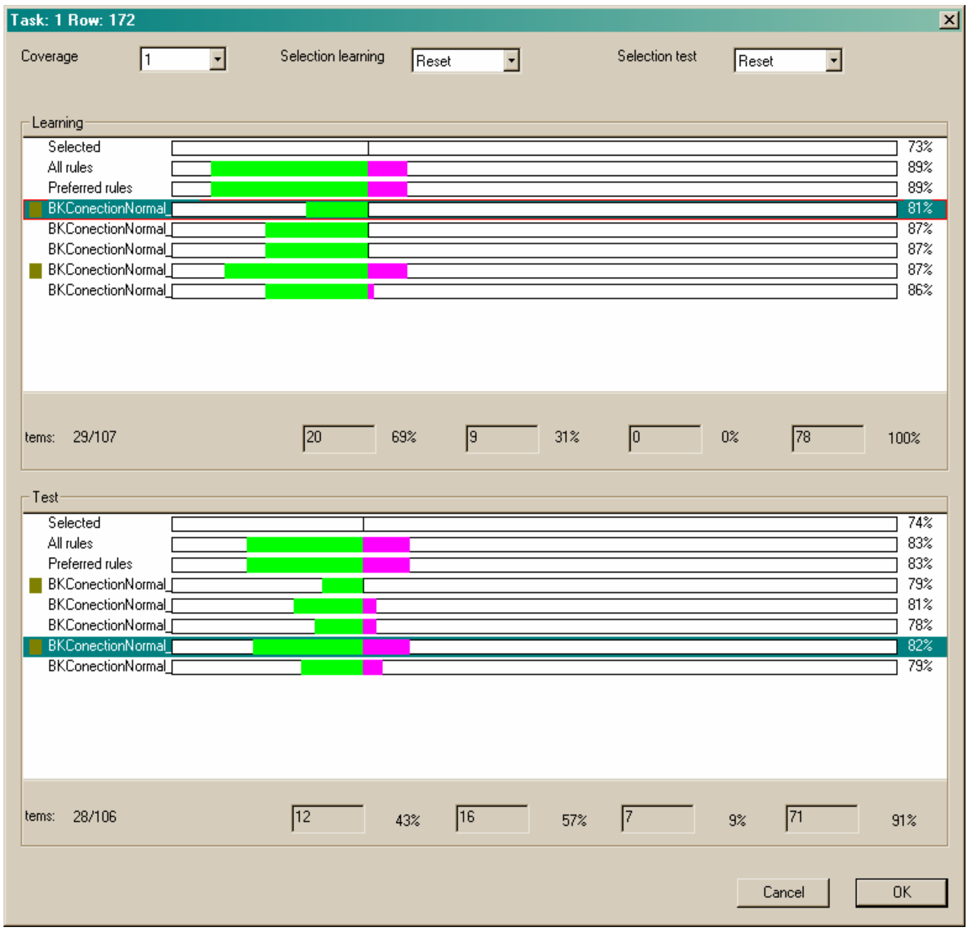
- Maximal length of rules – 6 (total number of predicates forming conjunction in rule premise);
- Coverage – 1.

Result: 5 rules:

```
BKConectionNormal_RN1=BKConectionNormal_N2
BKConectionNormal_RN2=BKConectionNormal_N3
BKConectionNormal_RN3=BKConectionNormal_N4
BKConectionNormal_RN4=BKConectionNormal_N5
BKConectionNormal_RN5=BKConectionNormal_N1
```

Validation of the rules overall coverage (see Fig.A.0)

Coverage is equal to 79% for the cases of class *Normal*.



```
Task: 1 Row: 172
Coverage 1        Selection learning Reset        Selection test Reset

Learning
Selected             73%
All rules            89%
Preferred rules      89%
BKConectionNormal_   81%
BKConectionNormal_   87%
BKConectionNormal_   87%
BKConectionNormal_   87%
BKConectionNormal_   86%
Items: 29/107    20  69%    9  31%    0  0%    78  100%

Test
Selected             74%
All rules            83%
Preferred rules      83%
BKConectionNormal_   79%
BKConectionNormal_   81%
BKConectionNormal_   78%
BKConectionNormal_   82%
BKConectionNormal_   79%
Items: 28/106    12  43%    16  57%    7  9%    71  91%
```

Fig.A.0.

Two rules are selected for use in BKConection*Normal* base classifier that are:

BKConection*Normal*_RN1 and BKConection*Normal*_RN4 .

Knowledge base of this classifier is as follows:

BKConection*Normal*_RN1 OR BKConection*Normal*_RN4 .

3. Training of the base classifier *BKPacketsNormal*

This base classifier deals with temporal data, each case corresponds to a connection and dataset is broken up in to samples: *Normal* and *Abnormal*.

The average number of packets comprising different attacks is 4,68 and for connections of the class Normal this length is 23,03.

The window used for analysis of connection status is 12 packets. According to the temporal data mining and decision making algorithms used in this project the input sequence is predicted up to the packet # 12. Just to remind, the algorithm is based of assessment of the value of discrepancy of input sequences approximated via use of a statistical model of the Normal connections.

The results of simulation given in Fig.A1 shows some attributes of distribution of such discrepancy for both *Normal* and *Abnormal* traffics given as a dependence from the number of the current packet within 12-packets window.

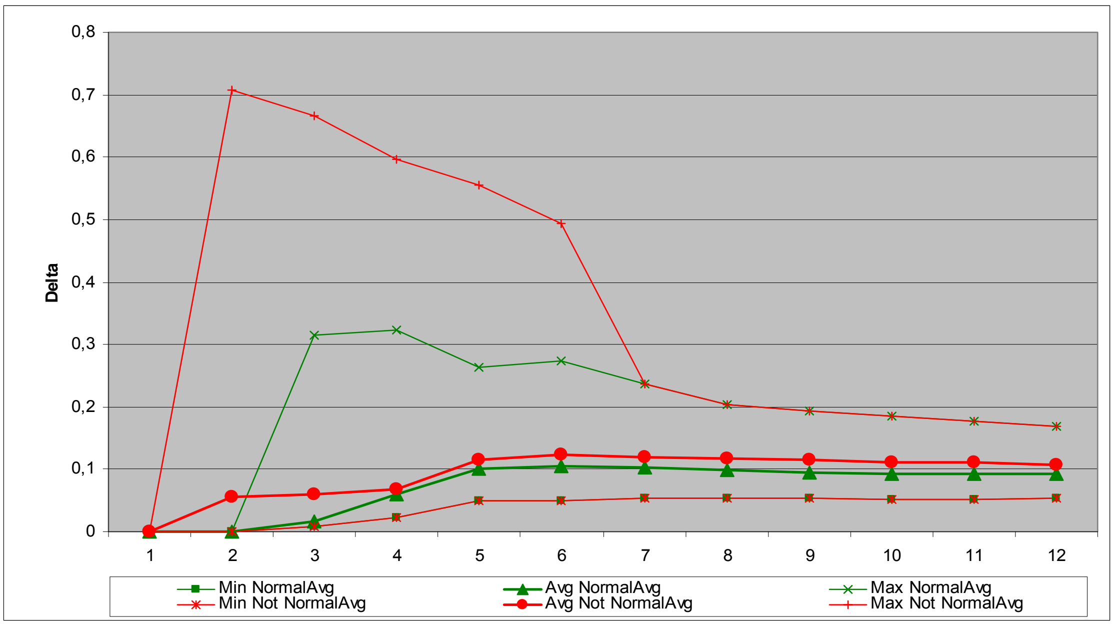


Fig.A1. Distribution of the discrepancy for both *Normal* and *Abnormal* traffics in time (given as a dependence from the number of the current packet within 12-packets window)

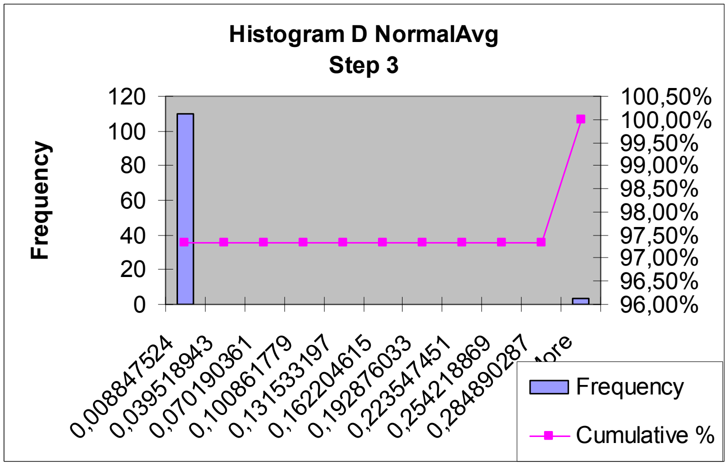


Fig.A2. Histogram of discrepancies for the packet number 3

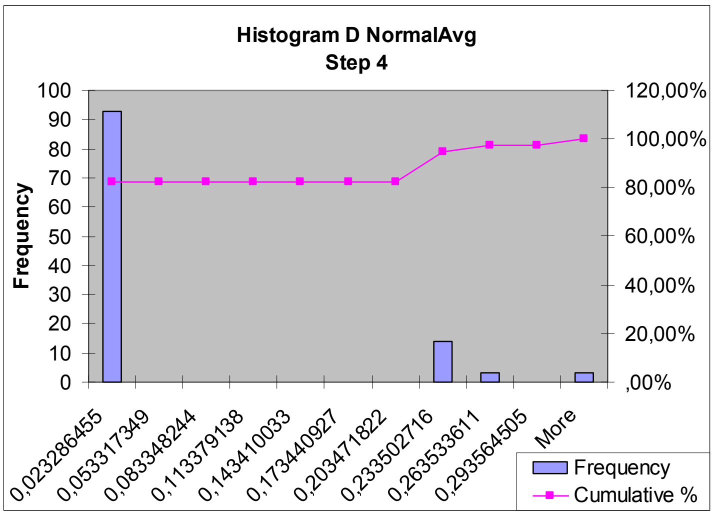


Fig.A3. Histogram of discrepancies for the packet number 4

The histograms presenting distributions of the above discrepancy for different values of the packet numbers (particularly, for packets of numbers 3, 4, 12) are given in Fig.A2, Fig.A3 and Fig.A4.

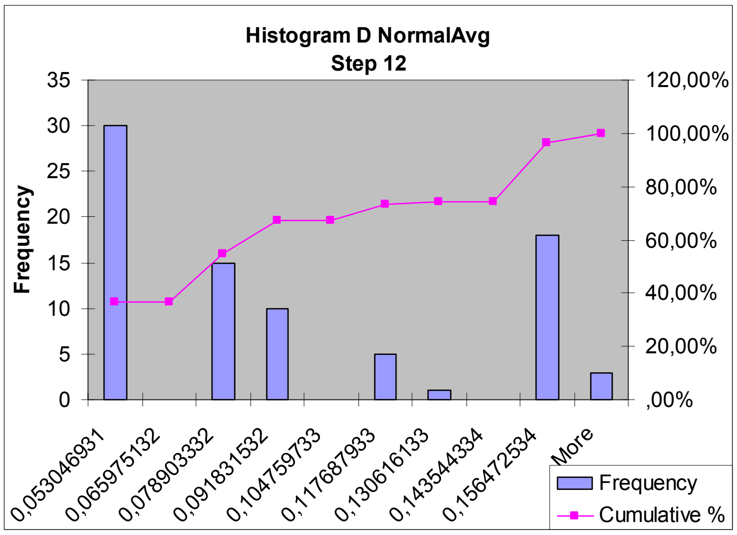


Fig.A4. Histogram of discrepancies for the packet number 12

3.1. Selection the threshold functions

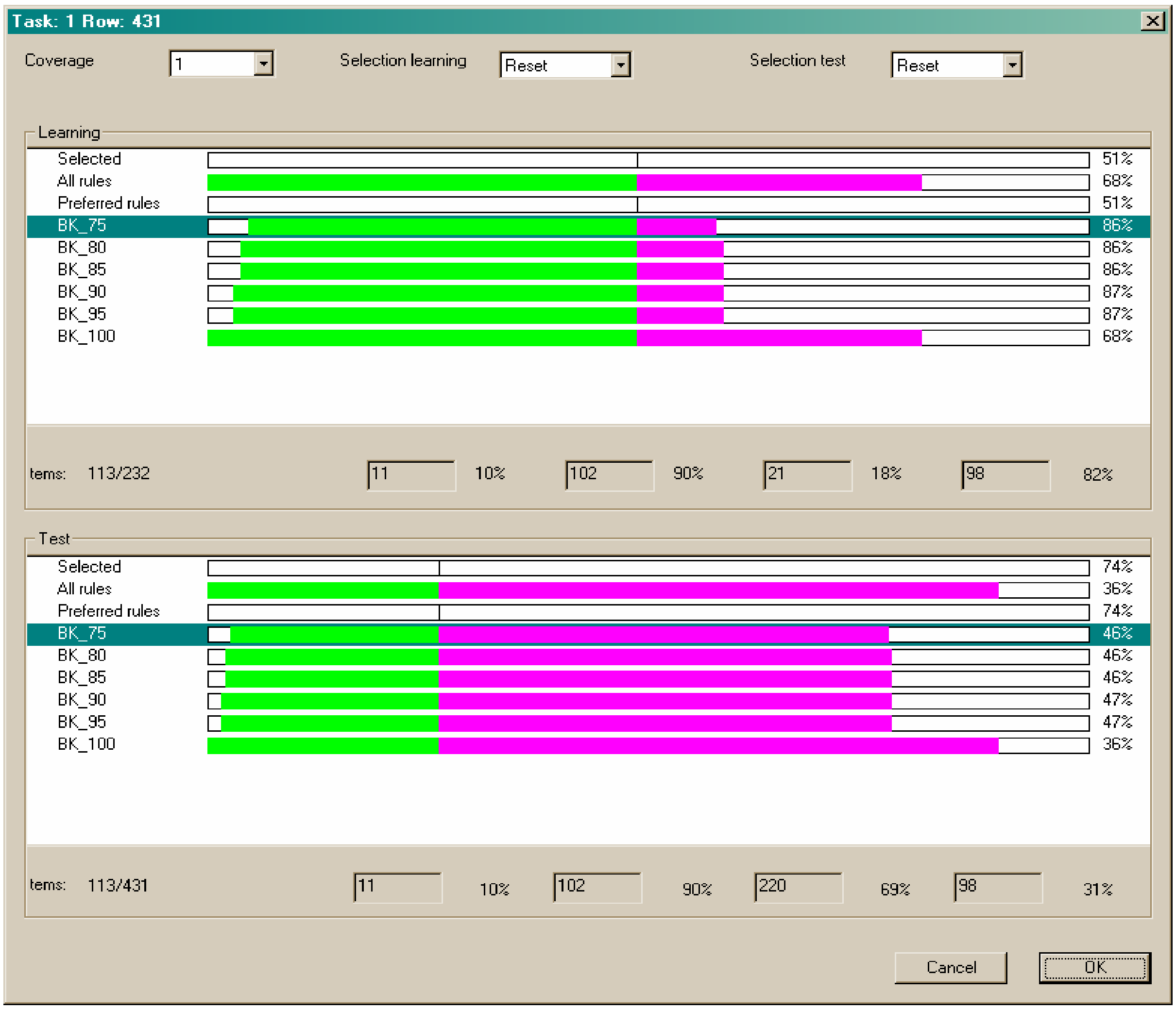
Several variants of the base classifier's algorithm were computed based on use of different threshold functions for each particular number of input packets within the chosen 12-packets window. The threshold functions were computed automatically for different given values of coverage factor. Each variant makes it possible to determine the probabilities comprising confusion matrix and that is why to select the value of the threshold that meets given constraints to false positive and false negative. Particularly, the following values of coverage factor for dataset of the class *Normal* were used in the above search procedures:

BK_75 – threshold= 75%
BK_80 – threshold= 80%
BK_85 – threshold= 85%
BK_90 – threshold= 90%
BK_95 – threshold= 95%
BK_100 – threshold= 100%

3.2. Assessment of the quality of performance of the base classifier given threshold function

The first sample which attributes are given in the Fig.A5 corresponds to the dataset of the class *Normal* whereas the second one corresponds to the dataset of class *Abnormal* which don't includes the cases of attack of type *SYNFlood* ("the length" of this attack is equal to 1 packet within a connection and this packet is not discernable as compared with normal connection; such kind of attack cannot be detected on the basis packet sequence it correspond to). The second sample (see Fig.A5) contains all connections including those ones which correspond to SYNFlood attack).

The base classifier selected is that which possess the best quality for threshold= 95% (see Fig.A5).



Task: 1 Row: 431

Coverage: 1 | Selection learning: Reset | Selection test: Reset

Learning

Selected	51%
All rules	68%
Preferred rules	51%
BK_75	86%
BK_80	86%
BK_85	86%
BK_90	87%
BK_95	87%
BK_100	68%

tems: 113/232 | 11 | 10% | 102 | 90% | 21 | 18% | 98 | 82%

Test

Selected	74%
All rules	36%
Preferred rules	74%
BK_75	46%
BK_80	46%
BK_85	46%
BK_90	47%
BK_95	47%
BK_100	36%

tems: 113/431 | 11 | 10% | 102 | 90% | 220 | 69% | 98 | 31%

Fig.A5.

4. Training of the base classifier *BK_5sec_Normal*

The training data found out such that it does not contain cases of attacks of classes S*YNFlood* and *PipeUpAdmin* (these attacks are developing in time very rapidly and that is why, as a rule, at the time of the 5 sec. interval end this kind of attack has already ended thus giving no information for making decision at that time). Thus, dataset for training this base classifier include connections of the class *Normal* and two types of attacks because the other attacks do not reveal themselves in such kind of data.

4.1. Analysis of datasets in the source *NetLevel – Agreg5sec*

Total number of cases – 224.

ID_ENTITY	224
Normal	112
Abnormal	112

Assignment of data for training of base classifier:

((Agreg5sec. ID_ENTITY)<='21:27:54,516691') AND *Normal*) OR ((Agreg5sec.ID_ENTITY <= '21:47:48,258865') AND *Abnormal*)

Number of cases of the classes:

Normal - 56
Abnormal – 56

Assignment of data for testing of base classifier:

((Agreg5sec. ID_ENTITY)>'21:27:54,516691') AND *Normal*) OR ((Agreg5sec.ID_ENTITY > '21:47:48,258865') AND *Abnormal*)

Number of cases of the classes:

Normal - 56
Abnormal – 56

4.2. Extraction rules of class *Normal*

Search for additional predicates on the basis of VAM algorithm

Result: 7 predicates as follows:

```
BKAgreg5secNormal_N1= (-0.6852682709941311*Agreg5sec_count_dest-
0.7282907364292875*Agreg5sec_count_serv_dest+3.6823386836223091 > 0) AND
(-0.5343733474352195*Agreg5sec_count_src-
0.8369532371383244*Agreg5sec_count_dest-
0.0604681322555463*Agreg5sec_count_serv_dest+2.2797493137921476 > 0)

BKAgreg5secNormal_N2=(-
0.9433238313802486*Agreg5sec_count_src+0.3318736945738371*Agreg5sec_count
_dest+1.6569773535293761 > 0)AND
(+0.4436340405383322*Agreg5sec_count_src-
0.8962080328113743*Agreg5sec_count_dest+2.5614289435551139 > 0)

BKAgreg5secNormal_N3= (-0.6201867455880064*Agreg5sec_count_dest-
0.7844542055448218*Agreg5sec_count_serv_src+3.5582341021334010 > 0)

BKAgreg5secNormal_N4= (-0.6547940947872492*Agreg5sec_count_src-
0.7558073123698573*Agreg5sec_count_serv_dest+3.1337185444345641 > 0)

BKAgreg5secNormal_N5= (-0.2523670376261696*Agreg5sec_count_src-
0.9676315819152408*Agreg5sec_count_serv_src+1.7864319565607989 > 0)

BKAgreg5secNormal_N6= (-0.5343733474352195*Agreg5sec_count_src-
0.8369532371383244*Agreg5sec_count_dest-
0.0604681322555463*Agreg5sec_count_serv_dest+2.2797493137921476 > 0)

BKAgreg5secNormal_N7= (-47.4270890788748010*Agreg5sec_count_src-
46.8011669194704820*Agreg5sec_count_dest-
49.1082675468836240*Agreg5sec_count_serv_dest+470.5431037702243200 > 0)
```

Extraction rules via use of GK2 algorithm:

While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative.

GK2 algorithm attributes:

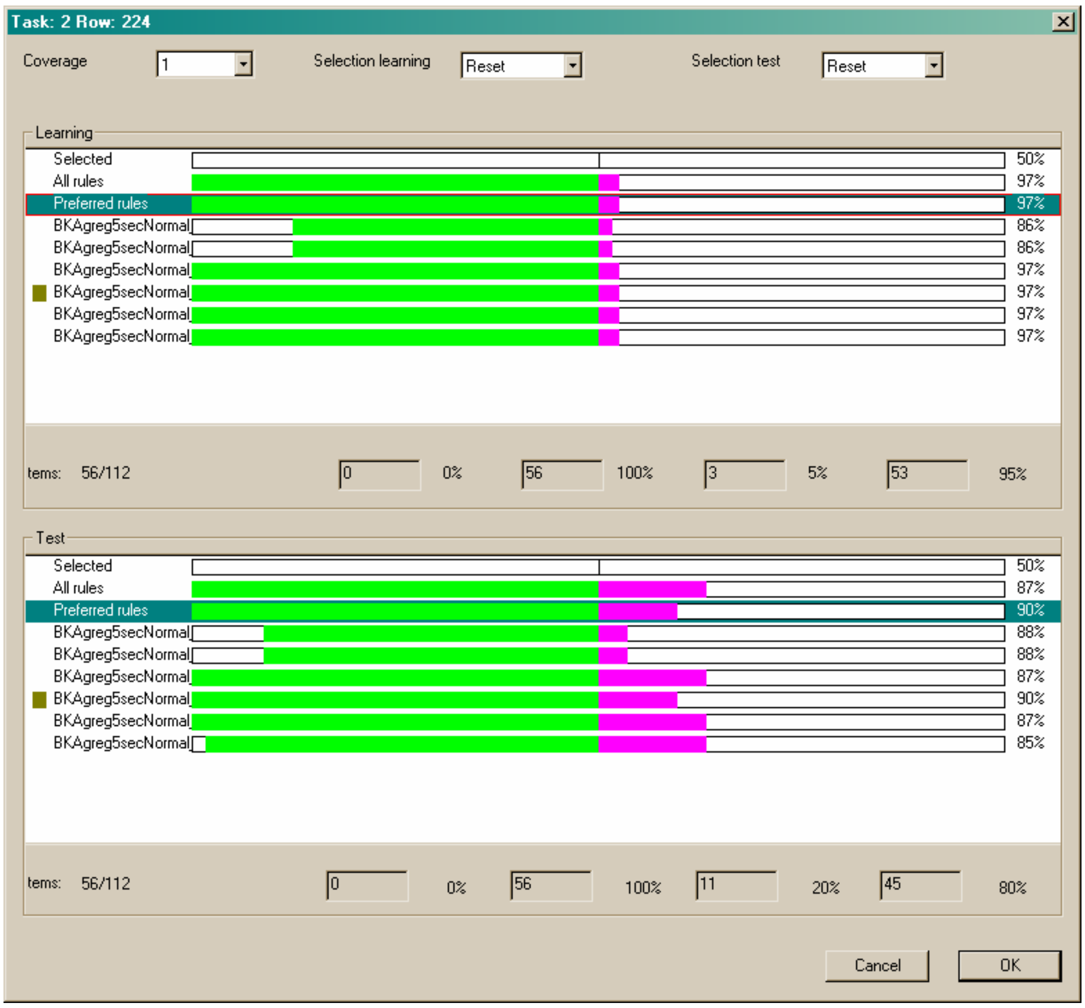
- Maximal "length" of the rules– 7 (total number of predicates connected by conjunction in premise);
- Coverage – 1

Result: 6 rules:

```
BKAgreg5secNormal_RN1=BKAgreg5secNormal_N6
BKAgreg5secNormal_RN2=BKAgreg5secNormal_N1
BKAgreg5secNormal_RN3=BKAgreg5secNormal_N2 AND BKAgreg5secNormal_N7
BKAgreg5secNormal_RN4=BKAgreg5secNormal_N2 AND BKAgreg5secNormal_N4
BKAgreg5secNormal_RN5=BKAgreg5secNormal_N2 AND BKAgreg5secNormal_N3
BKAgreg5secNormal_RN6=BKAgreg5secNormal_N2 AND BKAgreg5secNormal_N5
```

Assessment of the quality of the rules extracted

The total coverage factor for all rules extracted for dataset of the class *Normal* – 100% (see Fig.A6 for details).



```
Task: 2 Row: 224

Coverage  1          Selection learning  Reset          Selection test  Reset

Learning
Selected             50%
All rules            97%
Preferred rules      97%
BKAgreg5secNormal    86%
BKAgreg5secNormal    86%
BKAgreg5secNormal    97%
BKAgreg5secNormal    97%
BKAgreg5secNormal    97%
BKAgreg5secNormal    97%

tems:  56/112     0   0%     56   100%     3   5%     53   95%

Test
Selected             50%
All rules            87%
Preferred rules      90%
BKAgreg5secNormal    88%
BKAgreg5secNormal    88%
BKAgreg5secNormal    87%
BKAgreg5secNormal    90%
BKAgreg5secNormal    87%
BKAgreg5secNormal    85%

tems:  56/112     0   0%     56   100%     11   20%     45   80%
```

Fig.A6

The best rule in respect to testing dataset (it is selected as the base classifier rule):

BK Agreg5sec*Normal*_RN4.

5. Training of the base classifier *BK NetLevel – Agreg100con*

This data source is computed on the basis of all the cases of *Normal* and *Abnormal* connections but some anomalies do not reveal themselves (non-discernable with *Normal* class connections) in it.

5.1. Analysis of datasets in the source *NetLevel –Agreg100con*.

Total number of the cases – 332.

ID_ENTITY	332
Normal	15
Abnormal	317

Because of limited number of cases of the class Normal, all the cases are used both in training and testing procedures.

Conditions for selection of training data:

(*Normal*) OR ((Agreg100con.ID_ENTITY <= '21:48:23,480853') AND *Abnormal*)
OR ((Agreg100con.ID_ENTITY >= '22:01:34,845119') AND *Abnormal*)

Number of cases of the classes:

Normal - 15
Abnormal – 156

Assignment of data for testing of base classifier:

(*Normal*) OR (((Agreg5sec.ID_ENTITY > '21:48:23,480853') AND *Abnormal*) AND ((Agreg5sec.ID_ENTITY < '21:47:48,258865') AND *Abnormal*))

Number of cases of the classes:

Normal - 15
Abnormal – 161

5.2. Extraction rules of class 1 *Normal*

Search for additional predicates on the basis of VAM algorithm

Result: 6 rules (predicates) as follows:

```
BKAgreg100conNormal_N1= ( (-1.0*Agreg100con_count_dest+
85.6672250771610920 > 0)AND NOT(-0.9999978565374079*
Agreg100con_count_dest-0.0020704880076024*
Agreg100con_count_serv_dest1+82.6828606139623760 > 0)AND
NOT(+0.8660540915078701*Agreg100con_count_dest-0.4999503081131942*
Agreg100con_count_serv_dest1-39.8995280219891410 > 0)AND
(+0.6702207833765178*Agreg100con_count_dest+0.7421617758482088*Agreg100con
_count_serv_dest1-69.5947824793456110 > 0) )  OR ( (-
1.0*Agreg100con_count_dest +85.6672250771610920 > 0)AND NOT(-
0.9999978565374079*Agreg100con_count_dest-0.0020704880076024*
Agreg100con_count_serv_dest1+82.6828606139623760 > 0)AND
(+0.8660540915078701*Agreg100con_count_dest-0.4999503081131942*
Agreg100con_count_serv_dest1-39.8995280219891410 > 0)AND
NOT(+0.6702207833765178*Agreg100con_count_dest+0.7421617758482088*Agreg100
con_count_serv_dest1-69.5947824793456110 > 0) ) OR ((NOT ( (-
1.0*Agreg100con_count_dest+85.6672250771610920 > 0)AND NOT(-
0.9999978565374079*Agreg100con_count_dest-
0.0020704880076024*Agreg100con_count_serv_dest1+82.6828606139623760 >
0)AND NOT(+0.8660540915078701*Agreg100con_count_dest-
0.4999503081131942*Agreg100con_count_serv_dest1-39.8995280219891410 >
0)AND (+0.6702207833765178*Agreg100con_count_dest+
0.7421617758482088*Agreg100con_count_serv_dest1-69.5947824793456110 > 0) )
     OR ( (-1.0*Agreg100con_count_dest +85.6672250771610920 > 0)AND NOT(-
0.9999978565374079*Agreg100con_count_dest-
0.0020704880076024*Agreg100con_count_serv_dest1+82.6828606139623760 >
0)AND (+0.8660540915078701*Agreg100con_count_dest-
0.4999503081131942*Agreg100con_count_serv_dest1-39.8995280219891410 >
0)AND NOT(+0.670220783376517 8*Agreg100con_count_dest+
0.7421617758482088*Agreg100con_count_serv_dest1-69.5947824793456110 >
0) )) AND (+0.9997062945178824*Agreg100con_count_dest+
0.0242347828817369*Agreg100con_Duration-29.8661040827249590 > 0)AND (-
0.9697592558278408*Agreg100con_count_dest+
0.2440634871016806*Agreg100con_Duration-138.5638718097152700 > 0))

BKAgreg100conNormal_N2=(-0.5074448557840566*
Agreg100con_count_serv_dest1+0.8616842335439925*Agreg100con_Duration-
540.1887797977055900 > 0)

BKAgreg100conNormal_N3=(+0.9911934594955068*Agreg100con_count_dest+0.13242
17725803773*Agreg100con_Duration-101.272854921298 4600 > 0)AND (-
0.9986189191765833*Agreg100con_count_dest+0.0525381219933927
*Agreg100con_Duration+49.8709414482802980 > 0)AND
(+0.1187457411991160*Agreg100con_count_dest+0.9929246944995741*Agreg100con
_Duration-578.4997239314012600 > 0)

BKAgreg100conNormal_N4=(+0.5481350872495229*Agreg100con_count_src-
0.8363898170864814*Agreg100con_count_dest+50.1172271637927270 > 0)AND (-
0.7140075159528438*Agreg100con_count_src-
0.7001380343638313*Agreg100con_count_dest+94.1250796631528740 > 0)AND
NOT(-0.0080065900516841*Agreg100con_count_src-
0.9999679467441665*Agreg100con_count_dest+83.7628284144334430 > 0)

BKAgreg100conNormal_N5=(+0.4142582308379404*Agreg100con_count_serv_src1+0.
9101593916358935*Agreg100con_Duration-593.9597276776623900 > 0)AND (-
0.5548301631005419* Agreg100con_count_serv_src1+
0.8319636350909972*Agreg100con_Duration-527.2686026260485100 > 0)

BKAgreg100conNormal_N6=( (+0.5844387466391736*Agreg100con_count_src-
0.8114378296744809*Agreg100con_count_serv_dest1+25.9735995501971500 >
0)AND (-0.9959752854657177*Agreg100con_count_src-
0.0896282920816968*Agreg100con_count_serv_dest1+50.6194808068264380 >
0)AND NOT(-0.3846057289944088*Agreg100con_count_src
+0.9230809461930624*Agreg100con_count_serv_dest1-2.6532537564522283 >
0)AND (+0.9986096161275863*Agreg100con_count_src-
0.0527146523986899*Agreg100con_count_serv_dest1-34.7089776722383410 > 0))
OR ( NOT(+0.5844387466391736*Agreg100con_count_src-
0.8114378296744809*Agreg100con_count_serv_dest1+25.9735995501971500 >
0)AND NOT(-0.9959752854657177*Agreg100con_count_src-
0.0896282920816968*Agreg100con_count_serv_dest1+50.6194808068264380 >
0)AND (-0.3846057289944088*Agreg100con_count_src+
0.9230809461930624*Agreg100con_count_serv_dest1-2.6532537564522283 > 0)AND
(+0.9986096161275863*Agreg100con_count_src-
0.0527146523986899*Agreg100con_count_serv_dest1-34.7089776722383410 > 0) )
```

Search for rules on the basis of GK2 algorithm

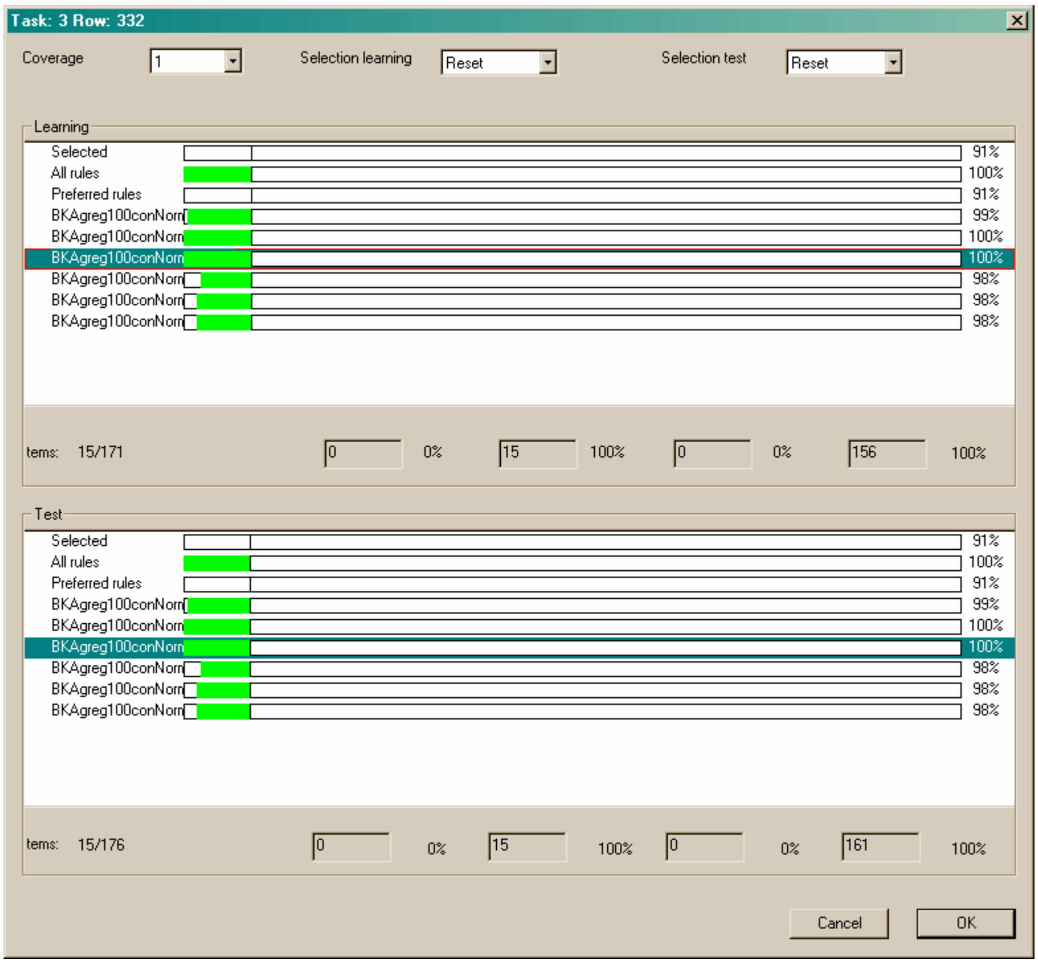
While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative.

Attributes:

- Maximal "length" of the rules– 7 (total number of predicates connected by conjunction in premise);
- Coverage – 1

Result: 9 rules as follows:

```
BKAgreg100conNormal_RN1=BKAgreg100conNormal_N1 AND
BKAgreg100conNormal_N2
BKAgreg100conNormal_RN2=BKAgreg100conNormal_N1 AND
BKAgreg100conNormal_N3
BKAgreg100conNormal_RN3=BKAgreg100conNormal_N1 AND
BKAgreg100conNormal_N5
BKAgreg100conNormal_RN4=BKAgreg100conNormal_N2 AND
BKAgreg100conNormal_N4
BKAgreg100conNormal_RN5=BKAgreg100conNormal_N3 AND
BKAgreg100conNormal_N4
BKAgreg100conNormal_RN6=BKAgreg100conNormal_N1 AND
BKAgreg100conNormal_N4
BKAgreg100conNormal_RN7=BKAgreg100conNormal_N4 AND
BKAgreg100conNormal_N5
BKAgreg100conNormal_RN8=BKAgreg100conNormal_N6
BKAgreg100conNormal_RN9=NOT BKAgreg100conNormal_N2 AND
BKAgreg100conNormal_N3
```


Task: 3 Row: 332

Coverage: 1 — Selection learning: Reset — Selection test: Reset

Learning

Selected	91%
All rules	100%
Preferred rules	91%
BKAgreg100conNorm	99%
BKAgreg100conNorm	100%
BKAgreg100conNorm	100%
BKAgreg100conNorm	98%
BKAgreg100conNorm	98%
BKAgreg100conNorm	98%

tems: 15/171 — 0 0% — 15 100% — 0 0% — 156 100%

Test

Selected	91%
All rules	100%
Preferred rules	91%
BKAgreg100conNorm	99%
BKAgreg100conNorm	100%
BKAgreg100conNorm	100%
BKAgreg100conNorm	98%
BKAgreg100conNorm	98%
BKAgreg100conNorm	98%

tems: 15/176 — 0 0% — 15 100% — 0 0% — 161 100%

Fig.A7

Assessment of the quality of the rules extracted

The total coverage factor for all rules extracted for dataset of the class *Normal* – 100% (see Fig.A7).

The basis rule selected as the best one:

BKAgreg100conNormal_RN3

6. Training of meta-classifier of the Network-based level

Training procedure for meta-classifier of the Network-based level is started for the class *Abnormal*.

While forming the meta-data, 3 additional features measured in logical scale containing information about type of an event initialized decision making procedure are added, in particular, they are as follows:

- InitConn – decision has to be made due to connection completion.
- Init5sec – decision has to be made due to completion of the 5-second interval;
- Init100conn – decision has to be made due to completion of the interval containing 100 connections.

6.1. Analysis of meta-data of the Network-based data source

Total number of cases used for computing meta-data – 2085.

ID_ENTITY	2085
Normal	1563
Abnormal	522

Break up of data into training and testing

Training data set:

Total number 1296 cases, between them:

Normal - 1034

Abnormal – 262

6.2. Training of the meta– classifier *MCNormal* (Meta-classifier of the Network-based data source)

Search for rules on the basis of GK2 algorithm

While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative.

Attributes:

- Maximal "length" of the rules– 7 (total number of predicates connected by conjunction in premise);
- Coverage – 1

The result: 18 rules given below:

```
MCNormal_RN1 = BK_ConnAgreg AND BK_Agreg5sec AND NOT Init100conn
MCNormal_RN2 = BK_ConnPacket AND BK_Agreg5sec AND NOT Init100conn
MCNormal_RN3 = BK_ConnAgreg AND BK_Agreg5sec AND InitConn
MCNormal_RN4 = BK_ConnPacket AND BK_Agreg5sec AND InitConn
MCNormal_RN5 = BK_ConnPacket AND  NOT BK_ConnAgreg AND BK_Agreg5sec
MCNormal_RN6 =  NOT BK_ConnPacket AND BK_ConnAgreg AND BK_Agreg5sec
MCNormal_RN7 = BK_Agreg100con AND Init5sec
MCNormal_RN8 = BK_ConnAgreg AND BK_Agreg100con
MCNormal_RN9 = BK_Agreg100con AND NOT InitConn
MCNormal_RN10 = BK_ConnPacket AND BK_Agreg100con
MCNormal_RN11 = BK_ConnAgreg AND Init5sec
MCNormal_RN12 = BK_ConnPacket AND BK_Agreg5sec AND Init5sec
MCNormal_RN13 = BK_ConnAgreg AND NOT InitConn AND NOT Init100conn
MCNormal_RN14 = BK_Agreg100con AND Init100conn
MCNormal_RN15 = NOT BK_ConnPacket AND NOT BK_Agreg5sec AND Init5sec
MCNormal_RN16 = NOT BK_ConnPacket AND NOT BK_Agreg5sec AND NOT InitConn
MCNormal_RN17 = BK_ConnAgreg AND NOT BK_Agreg5sec AND NOT InitConn
MCNormal_RN18 = NOT BK_ConnPacket AND BK_ConnAgreg AND NOT InitConn
```

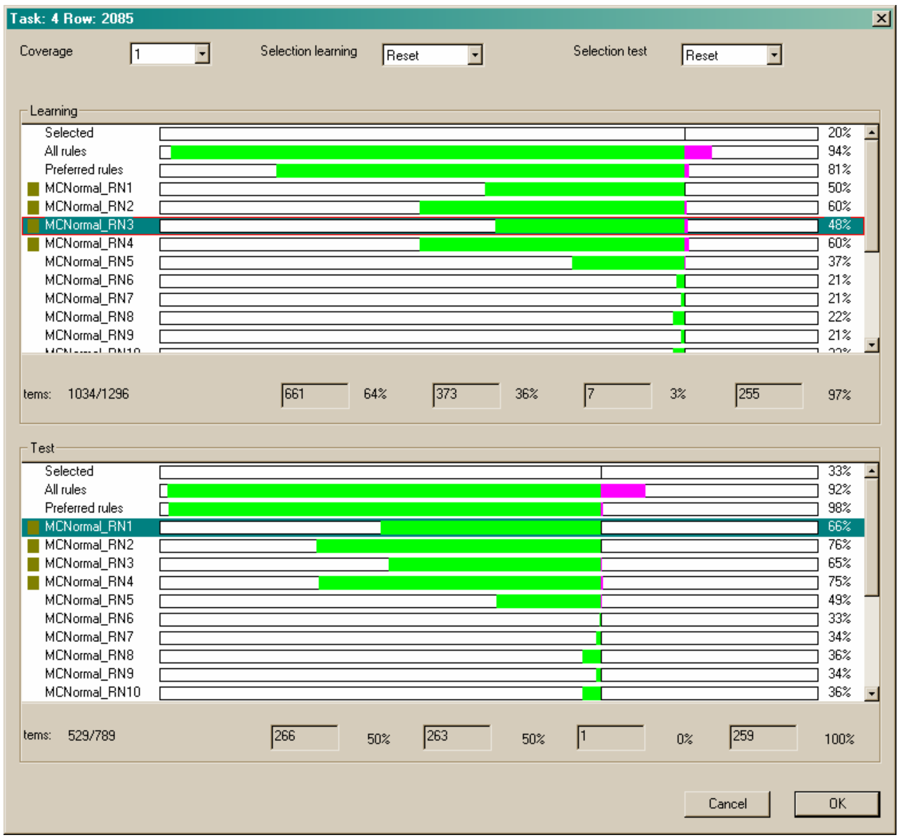
Assessment of the quality of the rules extracted

The total coverage factor for all rules extracted for dataset of the class *Normal* – 98% (see Fig.A8 for details).

The basis rule selected as the best ones assigned coverage factors are as follows:

```
MCNormal_RN1 – 66%
MCNormal_RN2 – 76%
MCNormal_RN3 – 65%
MCNormal_RN4 – 75%
```

Total coverage of the training data set – 78%
Total coverage of the testing data set 98%



```
Task: 4 Row: 2085
Coverage  1        Selection learning  Reset        Selection test  Reset

Learning
Selected          20%
All rules         94%
Preferred rules   81%
MCNormal_RN1      50%
MCNormal_RN2      60%
MCNormal_RN3      48%
MCNormal_RN4      60%
MCNormal_RN5      37%
MCNormal_RN6      21%
MCNormal_RN7      21%
MCNormal_RN8      22%
MCNormal_RN9      21%
tems:  1034/1296   661  64%   373  36%   7  3%   255  97%

Test
Selected          33%
All rules         92%
Preferred rules   98%
MCNormal_RN1      66%
MCNormal_RN2      76%
MCNormal_RN3      65%
MCNormal_RN4      75%
MCNormal_RN5      49%
MCNormal_RN6      33%
MCNormal_RN7      34%
MCNormal_RN8      36%
MCNormal_RN9      34%
MCNormal_RN10     36%
tems:  529/789   266  50%   263  50%   1  0%   259  100%
```

Fig.A.8

The resulting meta-classification rule is as follows:

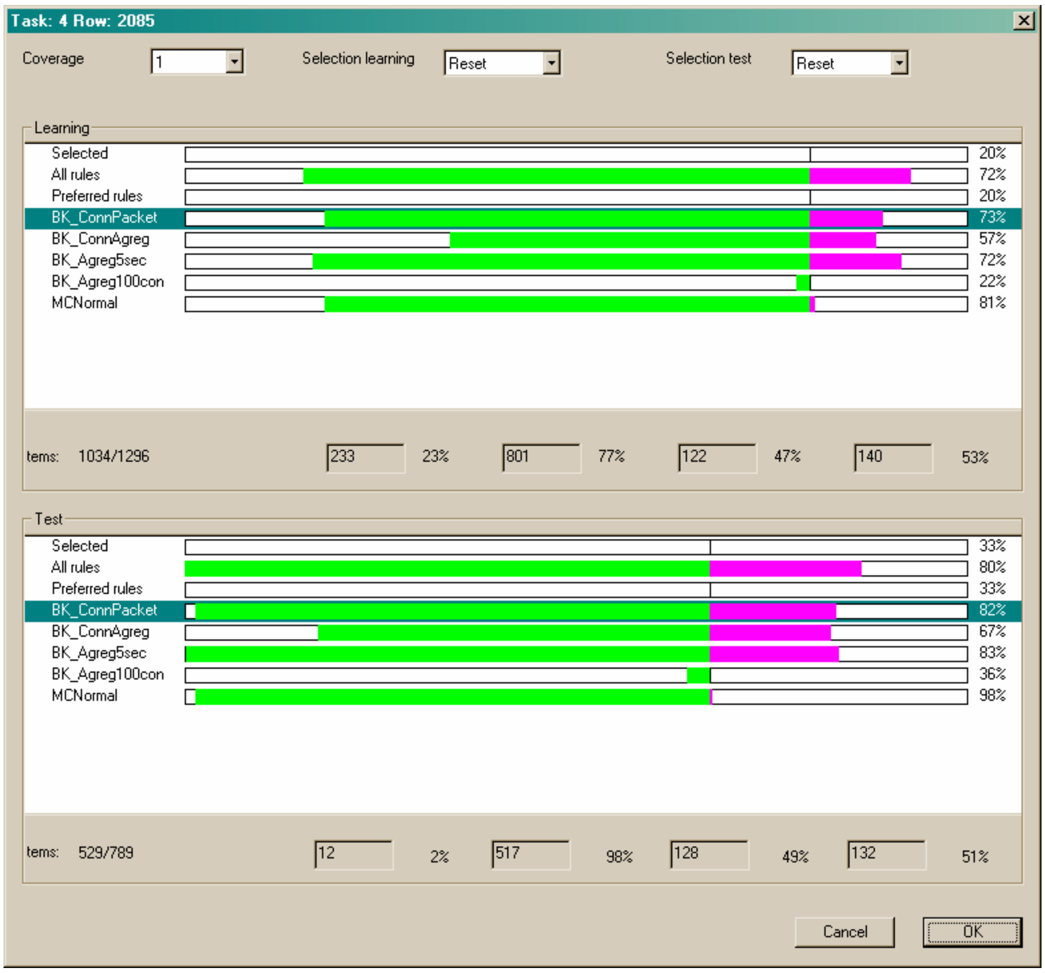
MC*Normal*: MC*Normal*_RN1 OR MC*Normal*_RN2 OR MC*Normal*_RN3 OR MC*Normal*_RN4

Let us assess the increase of quality provided by use of meta-classifier as compared with the best base classifier. The best base classifier provides 73% classification quality for the cases of the training dataset and 83% – for the cases of the testing dataset (see Fig.A9).

At the same time, the developed meta-classifier provides 81% classification quality for the cases of the training dataset and 98% – for the cases of the testing dataset.

Meta-classifier uses the following base classifiers of the Network-based level:

- BK_ConnAgreg;
- BK_ConnPacket;
- BK_Agreg5sec.



Coverage 1 Selection learning Reset Selection test Reset

Learning

Selected	20%
All rules	72%
Preferred rules	20%
BK_ConnPacket	73%
BK_ConnAgreg	57%
BK_Agreg5sec	72%
BK_Agreg100con	22%
MCNormal	81%

tems: 1034/1296 233 23% 801 77% 122 47% 140 53%

Test

Selected	33%
All rules	80%
Preferred rules	33%
BK_ConnPacket	82%
BK_ConnAgreg	67%
BK_Agreg5sec	83%
BK_Agreg100con	36%
MCNormal	98%

tems: 529/789 12 2% 517 98% 128 49% 132 51%

Fig. A.9

Appendix B. Data Sources of OS and Application Level

1. Learning detection of *FTPCrack* attack

Training and testing dataset used for learning are structured in the same way as dataset of the network-based level considered above in the Appendix A. The sequence of activities is also the same as previously described in Appendix A. This is the reason why the procedures below are described very briefly.

1.1. Learning of classification on the basis of temporal data (sequences of headers of packets)

Length of the sliding window is equal to 4 packets. Here it is also used a multi-dimensional regression for prognosis of the packets' fields on the basis of information has already been received.

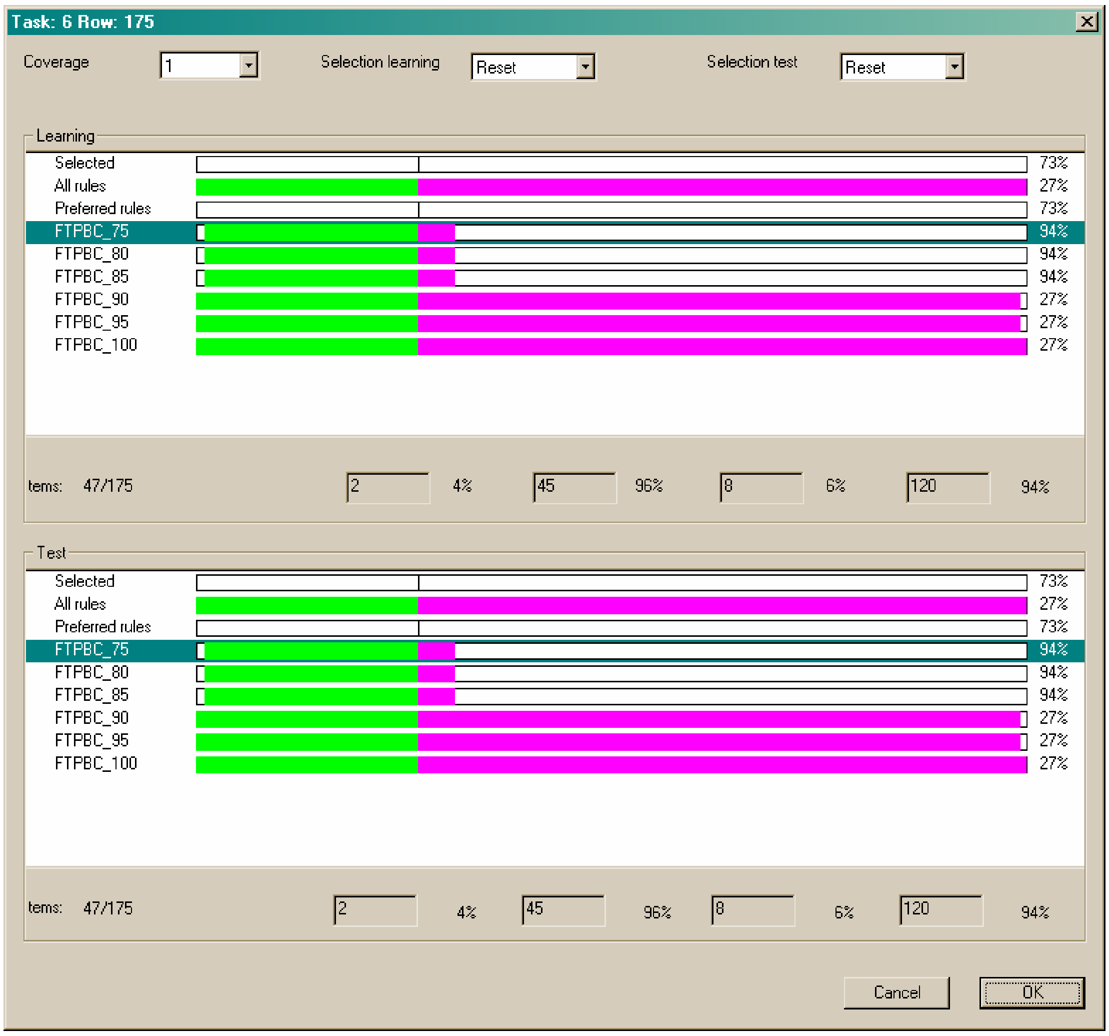
The model for assessment of the discrepancy between sequences of packet headers received and those corresponding model of traffic for *FTPCrack* attack is built. Histograms corresponding to distribution of discrepancy dependent of the number of input packet are computed.

1.2. Assessment of the quality of base classifier for different values of discrepancy threshold

Several variants of the base classifier's algorithm were computed based on use of different threshold functions for each particular number of input packets within the chosen 4-packets window. The threshold functions were computed automatically for different given values of coverage factor. Each a variant makes it possible to determine the probabilities comprising confusion matrix and that is why to select the value of the threshold that meets given constraints to false positive and false negative. Particularly, the following values of coverage factor for dataset of the class *Normal* were used in the above search procedures:

FTPBC_75 – threshold 75%
FTPBC_80 – threshold 80%
FTPBC_85 – threshold 85%
FTPBC_90 – threshold 90%
FTPBC_95 – threshold 95%
FTPBC_100 – threshold 100%

The result of selection is base classifier corresponding to the coverage threshold equal to 85% (see Fig. B1).


Task: 6 Row: 175

Coverage 1 | Selection learning Reset | Selection test Reset

Learning

Selected	73%
All rules	27%
Preferred rules	73%
FTPBC_75	94%
FTPBC_80	94%
FTPBC_85	94%
FTPBC_90	27%
FTPBC_95	27%
FTPBC_100	27%

tems: 47/175 2 4% 45 96% 8 6% 120 94%

Test

Selected	73%
All rules	27%
Preferred rules	73%
FTPBC_75	94%
FTPBC_80	94%
FTPBC_85	94%
FTPBC_90	27%
FTPBC_95	27%
FTPBC_100	27%

tems: 47/175 2 4% 45 96% 8 6% 120 94%

Fig.B.1

The coverage factor of this classifier is equal to 96%.

2. Training of the base classifier *FTPBC_Agreg5sec*

While analyzing data of training dataset, we determine the attribute having no variation (features) to delete from dataset as non–informative that is here "*hots*".

2.1. Learning of classification on the basis of temporal data (sequences of headers of packets)

It consists of search for additional predicates on the basis of *VAM* algorithm for the subspace *"failed logins"–"successful_logins"*

The result is as follows:

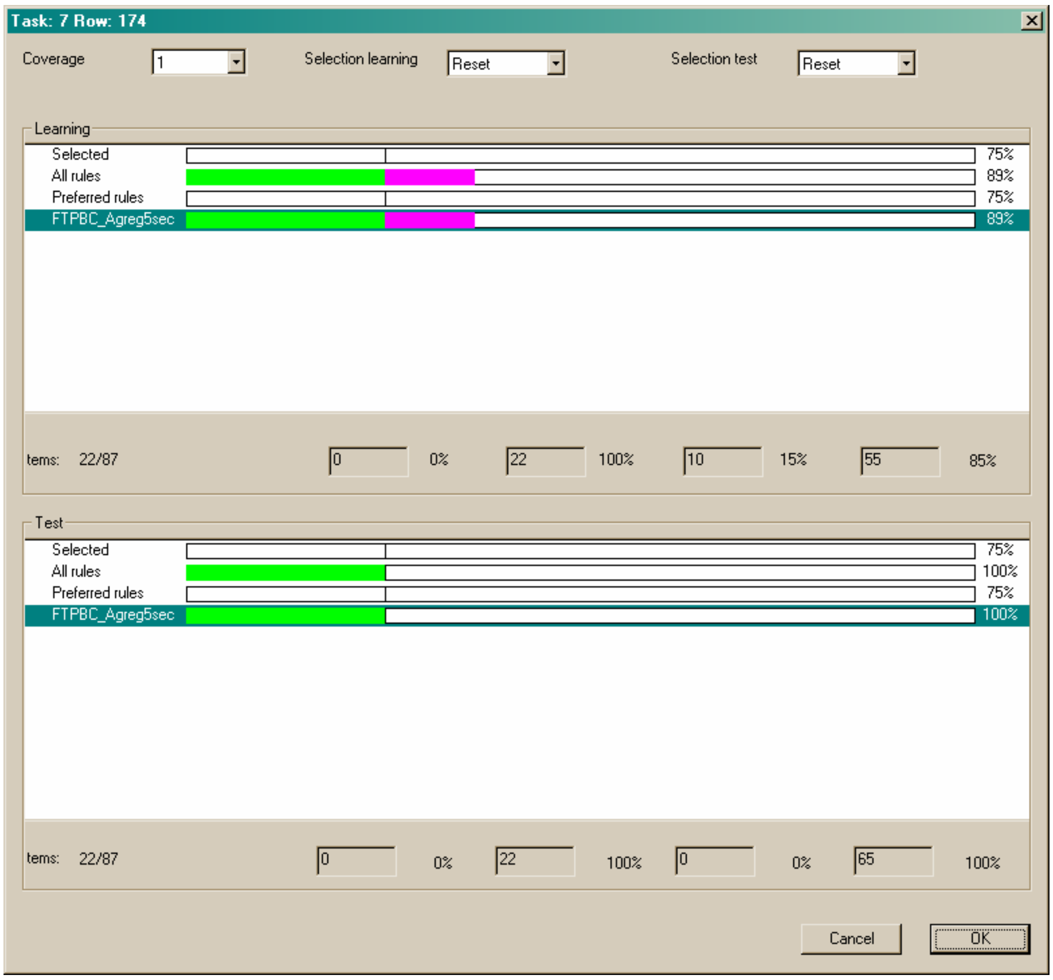
```
BCFTP_FTPCrackAgreg5sec=(+0.9674332962268508*failed_logins-
0.2531260898280744*successful_logins-0.8269450936460111 > 0)
```

Due to small dimensionality of this space this predicate is single and it is used as the premise of the respective classification rule.

2.2. Assessment of the quality of base classifier

The value of the coverage factor for this rule intended for detection of *FTPCrack* attack is equal to 100%.

The resulting probability of correct classification for this classifier (*FTPBC_Agreg5sec*) corresponds to 89% for training dataset and 100% for testing one (see Fig.B.2).


Task: 7 Row: 174

Coverage 1 Selection learning Reset Selection test Reset

Learning

Selected	75%
All rules	89%
Preferred rules	75%
FTPBC_Agreg5sec	89%

tems: 22/87 0 0% 22 100% 10 15% 55 85%

Test

Selected	75%
All rules	100%
Preferred rules	75%
FTPBC_Agreg5sec	100%

tems: 22/87 0 0% 22 100% 0 0% 65 100%

Fig.B.2

3. Training of the base classifier *FTPBC_Agreg30con*

This classifier makes decisions on the basis of an aggregation of the input information of the length 30 connections.

3.1. Dataset analysis

Total number of the instances – 49

ID_ENTITY	49
FTPCrack	16
Other	33

3.2. Extraction rules of class l: *FTPCrack*

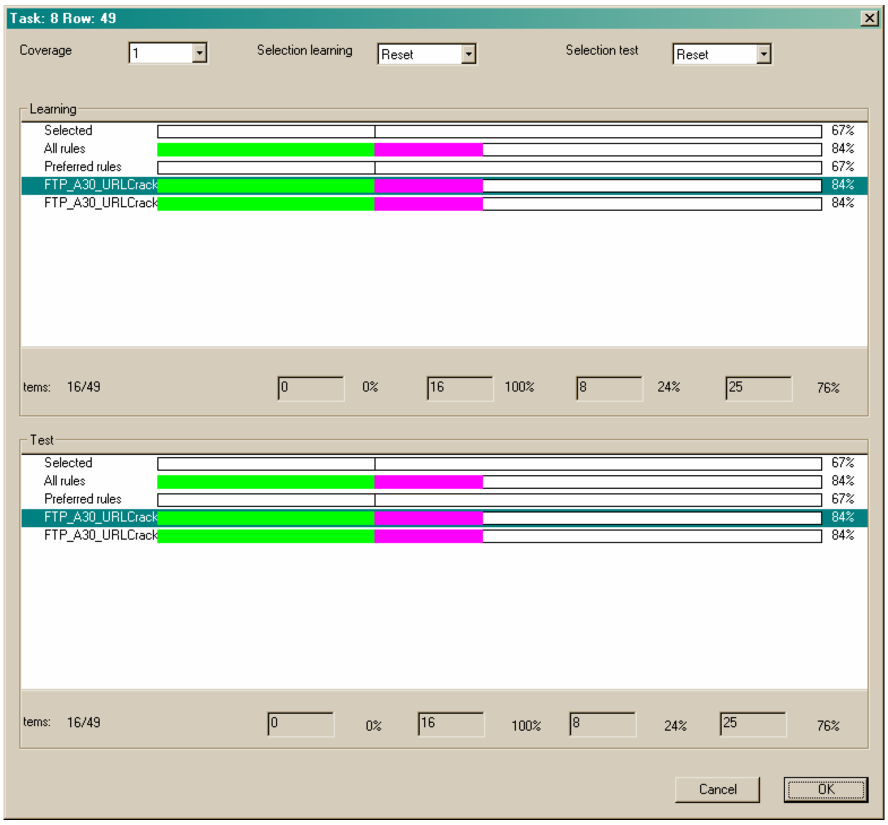
Search for additional predicates on the basis of *VAM* algorithm. It results in extraction of 2 predicates:

```
FTP_A30_FTPCrack1=(0.9990071650342662 * failed_logins +0.04454979472 *
Duration-28.9938983637157080>0) AND (0.9999994775514343* failed_logins+
0.0010222019656996*Duration -1.4399535579769640 > 0)
```

```
FTP_A30_FTPCrack2=(-0.9990125210414768*successful_logins +0.044429526
*Duration+1.0262718031633660 >0)AND (-0.9999995594442007*
successful_logins+0.0009386753456492* Duration+28.7101899404217950 > 0)
```
These predicates are used as the premises for the rules.

3.3. Assessment of the quality of the rules extracted

The total value of the coverage factor for these rules used for classification of the *FTPCrack* attacks is equal to 100%.



```
Task: 8 Row: 49
Coverage 1        Selection learning Reset        Selection test Reset

Learning
Selected                 67%
All rules                84%
Preferred rules          67%
FTP_A30_URLCrack         84%
FTP_A30_URLCrack         84%

tems: 16/49     0   0%     16   100%     8   24%     25   76%

Test
Selected                 67%
All rules                84%
Preferred rules          67%
FTP_A30_URLCrack         84%
FTP_A30_URLCrack         84%

tems: 16/49     0   0%     16   100%     8   24%     25   76%

Cancel   OK
```

Fig.B.3

The base rule of the classifier *FTPBC_Agreg30con* intended for detection of *FTPCrack* attack is the rule *FTP_A30_FTPCrack1* .

4. Training of the meta–classifier for detection of *FTPCrack* in the application level

4.1. Dataset analysis

While forming the meta-data, 3 extra features measured in logical scale containing information about type of an event initialized decision making procedure are added, in particular, they are as follows:

-InitConn – decision making procedure is initiated by event "*end of connection*";

- Init5sec - decision making procedure is initiated by event "*end of 5 second interval*":

- Init30conn – decision making procedure is initiated by event "*end of 30 connection length interval*".

The total number of instances in meta–dataset is equal to 275.

ID_ENTITY	275

FTPCrack	80
other	195

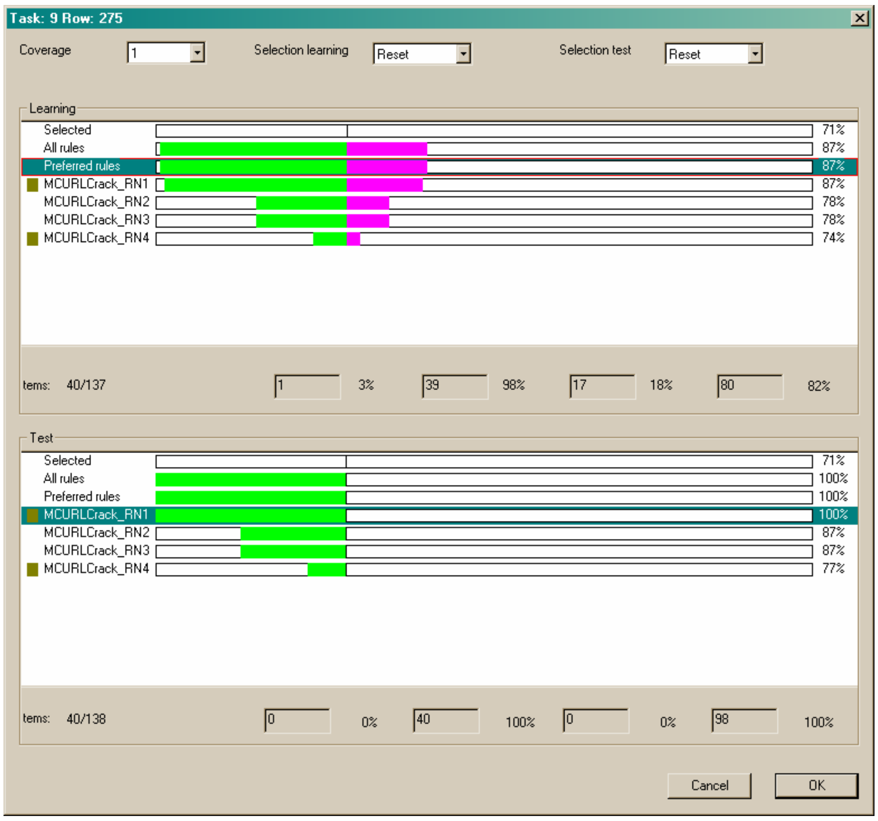
Breaking up the meta–dataset into training and testing ones:

Training dataset contains 137 instances, at that, distribution of classes:
of class *FTPCrack* - 40
other – 97

4.2. Extraction rules of class 1: *FTPCrack* forming meta– classifier *FTPMC_FTPCrack*

Search for rules on the basis of GK2 algorithm

While analyzing data of training dataset, we determine the attributes having no variation (features) to delete from dataset as non–informative.


Task: 9 Row: 275

Coverage: 1 — Selection learning: Reset — Selection test: Reset

Learning

Selected	71%
All rules	87%
Preferred rules	87%
MCURLCrack_RN1	87%
MCURLCrack_RN2	78%
MCURLCrack_RN3	78%
MCURLCrack_RN4	74%

tems: 40/137 — 1 3% — 39 98% — 17 18% — 80 82%

Test

Selected	71%
All rules	100%
Preferred rules	100%
MCURLCrack_RN1	100%
MCURLCrack_RN2	87%
MCURLCrack_RN3	87%
MCURLCrack_RN4	77%

tems: 40/138 — 0 0% — 40 100% — 0 0% — 98 100%

Fig.B.4

Rule's attributes:

- Maximal length of rules – 6 (the total number of predicates forming conjunction in rule premise)
- Coverage factor – 1

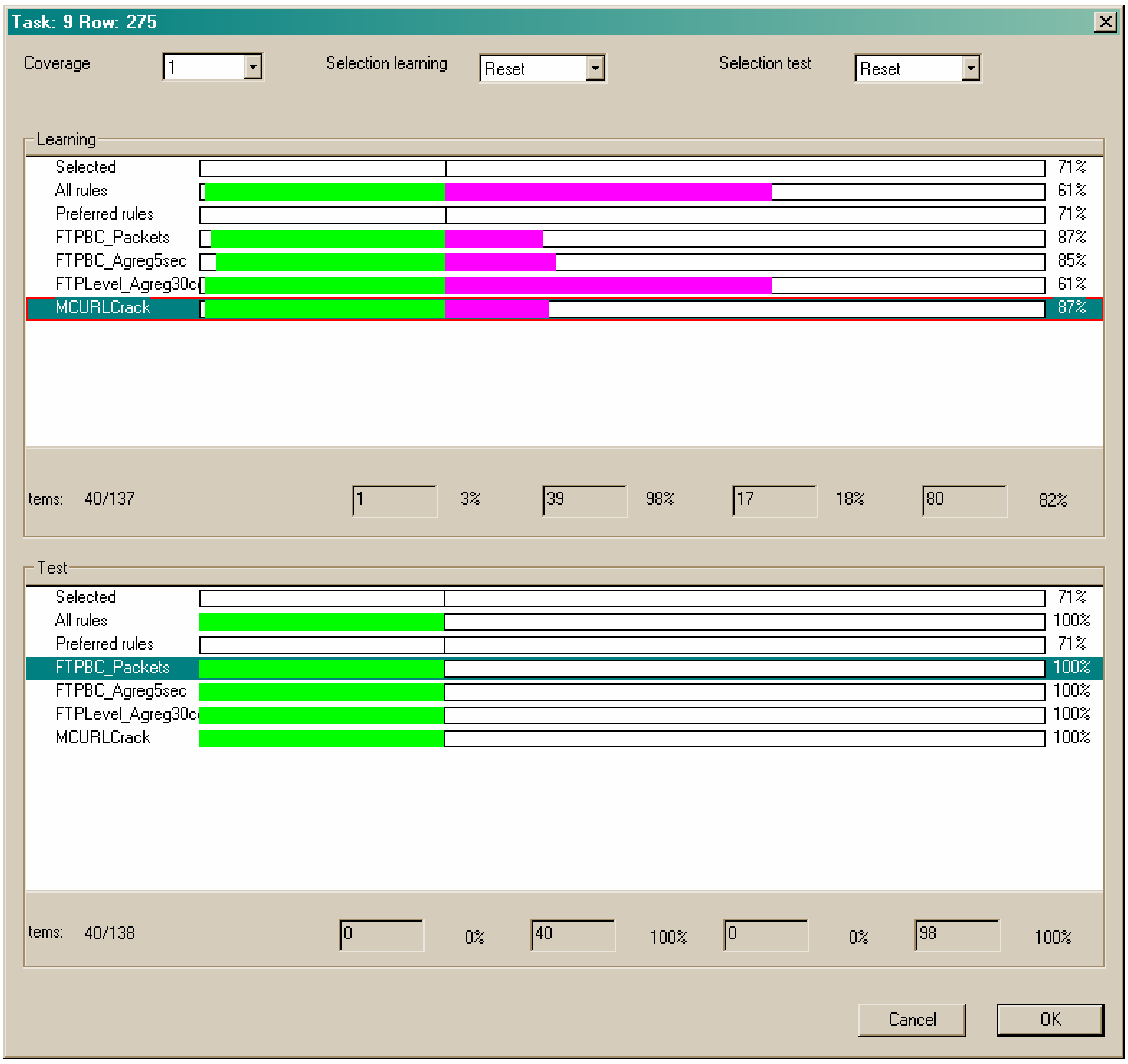
The result: 4 rules as follows:

```
MCFTPCrack_RN1 = FTPBC_Packets
MCFTPCrack_RN2 = FTPBC_Agreg5sec AND InitConn
MCFTPCrack_RN3 = FTPBC_Agreg5sec AND  NOT Init5sec
MCFTPCrack_RN4 = FTPBC_Agreg5sec AND Init30conn
```

The total coverage factor of these rules intended for detection of *FTPCrack* attack is equal to 98%.

The selected rules are those which provide the maximal coverage it both training and testing datasets:

```
MCFTPCrack_RN1 – 87%
MCFTPCrack_RN4 – 76%
```


Task: 9 Row: 275

Coverage 1 · Selection learning Reset · Selection test Reset

Learning

Selected	71%
All rules	61%
Preferred rules	71%
FTPBC_Packets	87%
FTPBC_Agreg5sec	85%
FTPLevel_Agreg30c	61%
MCURLCrack	87%

tems: 40/137 · 1 · 3% · 39 · 98% · 17 · 18% · 80 · 82%

Test

Selected	71%
All rules	100%
Preferred rules	71%
FTPBC_Packets	100%
FTPBC_Agreg5sec	100%
FTPLevel_Agreg30c	100%
MCURLCrack	100%

tems: 40/138 · 0 · 0% · 40 · 100% · 0 · 0% · 98 · 100%

Figure B.5

4.3. Assessment of the quality of the rules extracted

Coverage over training dataset – 98%
Coverage over testing dataset – 100%

The final rule according to which meta–classifier detects *FTPCrack* attack:

```
MCFTPCrack: MCFTPCrack_RN1 OR MCFTPCrack_RN4
```

Let us assess the quality of this meta-classifier as compared with the quality provided by particular base classifiers. The best base classifier provides the correct decision in 87% of cases over training dataset and in 95% over testing one. The designed meta-classifier provides 87% and 100% respectively having the value of coverage factor equal to 98%.